LUMINARY MICRO®

Stellaris® Graphics Library

USER'S GUIDE

PDL-GRL-uvision-UG-2523

Legal Disclaimers and Trademark Information

Luminary Micro, Inc.
108 Wild Basin, Suite 350
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com

Revision Information

This is version 2523 of this document, last updated on April 03, 2008.

Table of Contents

1 Introduction

The Luminary Micro® Stellaris® Graphics Library provides a set of graphics primitives and a widget set for creating graphical user interfaces on Stellaris microcontroller-based boards that have a graphical display. The graphics library consists of three layers (with each subsequent layer building upon the previous layer to provide more functionality):

- The display driver layer. This must be supplied by the application since it is specific to the display in use.
- The graphics primitives layer. This provides the ability to draw individual items on the display, such as lines, circles, text, and so on.
- The widget layer. This provides an encapsulation of one or more graphics primitives to draw a user interface element on the display, along with the ability to provide application-defined responses to user interaction with the element.

An application can call APIs in any of the three layers, which are non-exclusive (in other words, it is valid to use widgets and also directly use graphics primitives). The choice of the layer to use depends upon the needs and requirement of the application.

The capabilities and organization of the graphics library are governed by the following design goals:

- They are written entirely in C except where absolutely not possible.
- They are easy to understand.
- They are reasonably efficient in terms of memory and processor usage.
- They are as self-contained as possible.
- Where possible, computations that can be performed at compile time are done there instead of at run time.

Some consequences of these design goals are:

- The graphics primitives are not necessarily as efficient as they could be (from a code size and/or execution speed point of view). While the most efficient piece of code for drawing any graphics primitive would be written in assembly and custom tailored to the specific display in use, further size optimizations of the graphics primitives would make them more difficult to understand.
- The widget set provides "super-widgets" that have more capabilities than may be used in a particluar application. While it would be more efficient to have a widget that performed just what the application required, this would require each widget to become multiple widgets with the same basic function but a mixture of the capabilities. A specific-function widget can be derived from one of the existing widgets if required.
- The APIs have a means of removing all error checking code. Since the error checking is usually only useful during initial program development, it can be removed to improve code size and speed.

Source Code Overview

The following is an overview of the organization of the graphics library source code, along with references to where each portion is described in detail.

`canvas.c`	The source code for the canvas widget, which is described in chapter 5.
`canvas.h`	The header containing prototypes for the canvas widget.
`checkbox.c`	The source code for the check box widget, which is described in chapter 6.
`checkbox.h`	The header containing prototypes for the checkbox widget.
`circle.c`	The source code for the circle primitives, which are described in chapter 3.
`container.c`	The source code for the container widget, which is described in chapter 7.
`container.h`	The header containing prototypes for the container widget.
`context.c`	The source code for the drawing context, which is described in chapter 3.
`fonts/`	The source files containing the font structures for the fonts provided with the graphics library.
`ftrasterize/`	The source code for the ftrasterize program, which is described in chapter 10.
`grlib.Opt`	The Keil uVision project options file used for building the graphics library.
`grlib.Uv2`	The Keil uVision project file used for building the graphics library.
`grlib.h`	The header containing prototypes for the graphics primitives.
`image.c`	The source code for the image primitives, which are described in chapter 3.
`line.c`	The source code for the line primitives, which are described in chapter 3.
`offscr1bpp.c`	The source code for the 1 BPP off-screen buffer display driver, which is described in chapter 3.
`offscr4bpp.c`	The source code for the 4 BPP off-screen buffer display driver, which is described in chapter 3.
`offscr8bpp.c`	The source code for the 8 BPP off-screen buffer display driver, which is described in chapter 3.
`pnmtoc/`	The source code for the pnmtoc program, which is described in chapter 10.
`pushbutton.c`	The source code for the push button widget, which is described in chapter 8.
`pushbutton.h`	The header containing prototypes for the push button widget.
`radiobutton.c`	The source code for the radio button widget, which is described in chapter 9.
`radiobutton.h`	The header containing prototypes for the radio button widget.
`readme.txt`	A short file describing the highlights of the graphics library.

`rectangle.c`	The source code for the rectangle primitives, which are described in chapter 3.
`string.c`	The source code for the string primitives, which are described in chapter 3.
`widget.c`	The source code for the widget framework, which is described in chapter 4.
`widget.h`	The header containging prototypes for the widget framework.

2 Display Driver

2.1 Introduction

A display driver is used to allow the graphics library to interface with a particular display. It is responsible for dealing with the low level details of the display, including communicating with the display controller and understanding the commands required to make the display controller behave as required.

The display driver must provide two things; the set of routines required by the graphics library to draw onto the screen and a set of routines for performing display-dependent operatins. The display dependent operations will vary from display to display, but will include an initialization routine, and may include such things as backlight control and contrast control.

The routines required by the graphics library are organized into a structure that describes the display driver to the graphics library. The tDisplay structure contains these function pointers, along with the width and height of the screen. An instantiation of this structure should be supplied by the display driver, along with a prototype for that structure in a display driver-specific header file.

For some displays, it may be more efficient to draw into a buffer in local memory and copy the results to the screen after all drawing operations are complete. This is usually true of 4 BPP displays, where there are two pixels in each byte of display memory (where writing a single pixel would require a display memory read followed by a display memory write). In this case, the Flush() operation is used to indicate that the local display buffer should be copied to the display.

If the display driver uses a buffer in local memory, it may be advantageous for the display driver to track dirty rectangles (in other words, portions of the buffer that have been changed and therefore must be updated to the display) to accelerate the process of copying the local memory to the display memory. However, dirty rectangle tracking is optional, and is entirely the responsibility of the display driver if it is used (in other words, the graphics library will not provide dirty rectangle information to the display driver).

2.2 Definitions

Functions

- unsigned long ColorTranslate (void ∗pvDisplayData, unsigned long ulValue)
- void Flush (void ∗pvDisplayData)
- void LineDrawH (void ∗pvDisplayData, long lX1, long lX2, long lY, unsigned long ulValue)
- void LineDrawV (void ∗pvDisplayData, long lX, long lY1, long lY2, unsigned long ulValue)
- void PixelDraw (void ∗pvDisplayData, long lX, long lY, unsigned long ulValue)
- void PixelDrawMultiple (void ∗pvDisplayData, long lX, long lY, long lX0, long lCount, long lBPP, const unsigned char ∗pucData, const unsigned char ∗pucPalette)
- void RectFill (void ∗pvDisplayData, const tRectangle ∗pRect, unsigned long ulValue)

2.2.1 Function Documentation

2.2.1.1 ColorTranslate

Translates a 24-bit RGB color to a display driver-specific color.

Prototype:
```
unsigned long
ColorTranslate(void *pvDisplayData,
               unsigned long ulValue)
```

Parameters:
pvDisplayData is a pointer to the driver-specific data for this display driver.
ulValue is the 24-bit RGB color. The least-significant byte is the blue channel, the next byte is the green channel, and the third byte is the red channel.

Description:
This function translates a 24-bit RGB color into a value that can be written into the display's frame buffer in order to reproduce that color, or the closest possible approximation of that color.

Returns:
Returns the display-driver specific color.

2.2.1.2 Flush

Flushes any cached drawing operations.

Prototype:
```
void
Flush(void *pvDisplayData)
```

Parameters:
pvDisplayData is a pointer to the driver-specific data for this display driver.

Description:
This functions flushes any cached drawing operations to the display. This is useful when a local frame buffer is used for drawing operations, and the flush would copy the local frame buffer to the display. If there are no cached operations possible for a display driver, this function can be empty.

Returns:
None.

2.2.1.3 LineDrawH

Draws a horizontal line.

Prototype:
```
void
LineDrawH(void *pvDisplayData,
          long lX1,
          long lX2,
          long lY,
          unsigned long ulValue)
```

Parameters:

pvDisplayData is a pointer to the driver-specific data for this display driver.

lX1 is the X coordinate of the start of the line.

lX2 is the X coordinate of the end of the line.

lY is the Y coordinate of the line.

ulValue is the color of the line.

Description:

This function draws a horizontal line on the display. The coordinates of the line are assumed to be within the extents of the display.

Returns:

None.

2.2.1.4 LineDrawV

Draws a vertical line.

Prototype:

```
void
LineDrawV(void *pvDisplayData,
          long lX,
          long lY1,
          long lY2,
          unsigned long ulValue)
```

Parameters:

pvDisplayData is a pointer to the driver-specific data for this display driver.

lX is the X coordinate of the line.

lY1 is the Y coordinate of the start of the line.

lY2 is the Y coordinate of the end of the line.

ulValue is the color of the line.

Description:

This function draws a vertical line on the display. The coordinates of the line are assumed to be within the extents of the display.

Returns:

None.

2.2.1.5 PixelDraw

Draws a pixel on the screen.

Prototype:

```
void
PixelDraw(void *pvDisplayData,
          long lX,
          long lY,
          unsigned long ulValue)
```

Parameters:

pvDisplayData is a pointer to the driver-specific data for this display driver.

lX is the X coordinate of the pixel.

lY is the Y coordinate of the pixel.

ulValue is the color of the pixel.

Description:

This function sets the given pixel to a particular color. The coordinates of the pixel are assumed to be within the extents of the display.

Returns:

None.

2.2.1.6 PixelDrawMultiple

Draws a horizontal sequence of pixels on the screen.

Prototype:

```
void
PixelDrawMultiple(void *pvDisplayData,
                  long lX,
                  long lY,
                  long lX0,
                  long lCount,
                  long lBPP,
                  const unsigned char *pucData,
                  const unsigned char *pucPalette)
```

Parameters:

pvDisplayData is a pointer to the driver-specific data for this display driver.

lX is the X coordinate of the first pixel.

lY is the Y coordinate of the first pixel.

lX0 is sub-pixel offset within the pixel data, which is valid for 1 or 4 bit per pixel formats.

lCount is the number of pixels to draw.

lBPP is the number of bits per pixel; must be 1, 4, or 8.

pucData is a pointer to the pixel data. For 1 and 4 bit per pixel formats, the most significant bit(s) represent the left-most pixel.

pucPalette is a pointer to the palette used to draw the pixels.

Description:

This function draws a horizontal sequence of pixels on the screen, using the supplied palette. For 1 bit per pixel format, the palette contains pre-translated colors; for 4 and 8 bit per pixel formats, the palette contains 24-bit RGB values that must be translated before being written to the display.

Returns:
None.

2.2.1.7 RectFill

Fills a rectangle.

Prototype:
```
void
RectFill(void *pvDisplayData,
         const tRectangle *pRect,
         unsigned long ulValue)
```

Parameters:
pvDisplayData is a pointer to the driver-specific data for this display driver.
pRect is a pointer to the structure describing the rectangle.
ulValue is the color of the rectangle.

Description:
This function fills a rectangle on the display. The coordinates of the rectangle are assumed to be within the extents of the display, and the rectangle specification is fully inclusive (in other words, both sXMin and sXMax are drawn, along with sYMin and sYMax).

Returns:
None.

3 Graphics Primitives

3.1 Introduction

The graphics primitives provide a set of low level drawing operations. These operations include drawing lines, circles, text, and bitmap images. A means of drawing into an off-screen buffer is also available, allowing multiple drawing operations to be performed into an off-screen buffer and the final result copied to the screen at once (which will eliminate flickering if a complex display with many overlapping entities is utilized).

Color is represented as 24-bit RGB, regardless of the display in use. The display driver will translate the provided 24-bit RGB value into the closest approximation available; this may be "on" and "off" for a monochrome display, a 16-bit 5-6-5 RGB color for a color display, or many other possible translations.

A display driver may draw into a local buffer and then copy the final results to the display once complete (this is typically the case for display formats that have more than one pixel in a byte since it is more efficient to store the display data in SRAM than to read from the display itself). GrFlush() is used to indicate that the drawing operations are complete and that the screen should be updated. It is important to call GrFlush() to ensure that all drawing operations have been refllected onto the screen.

A large set of fonts are provided, based on the Computer Modern typeface defined by Professor Donald E. Knuth for the TEX typesetting system. Serif and san-serif fonts are provided in regular, bold, and italic styles in even sizes from 12 to 48 point, and a small caps font is provided in even sizes from 12 to 48 point. Additionally, a custom designed 6x8 fixed-point is provided. The fonts are stored in the `grlib/fonts` directory, which each font residing in its own file.

Note:
The Computer Modern typeface does not provide glyphs for the "<", ">", "\", "^", "_", "{", "|", "}", or "~" characters, instead providing different glyphs for these characters.

3.1.1 Drawing Context

All drawing operations take place in terms of a drawing context. This context describes the display to which the operation occurs, the color and font to use, and the region of the display to which the operation should be limited (the clipping region).

A drawing context must be initialized with GrContextInit() before it can be used to perform drawing operations. The other GrContext...() functions are used to modify the properties of the drawing context.

The colors in a drawing context are typically set with GrContextForegroundSet() and GrContextBackgroundSet(), which utilizes the color translation capabilities of the context's display driver to convert the 24-bit color to a display-appropriate value. For applications that frequently switch between colors, it may be more efficient to use DpyColorTranslate() to perform the 24-bit RGB color translation once per color and then use GrContextForegroundSetTranslated() and GrContextBackgroundSetTranslated() to change between colors.

3.1.2 Off-screen Buffers

Off-screen buffers provide a mechanism for drawing into a buffer in memory instead of directly to the screen. The resulting buffer is in the Graphics Library image format, so the off-screen buffer can be copied to a display at any time using GrImageDraw().

The off-screen buffers appear as separate display drivers that draw into memory. Unlike a normal display driver that draws into memory, performing a flush on an off-screen buffer does nothing.

Off-screen buffer display drivers are provided in 1 bit-per-pixel (BPP) format, 4 BPP format, and 8 BPP format. The image buffers that they produce are always in uncompressed format (since it would be far too computationally expensive to try to keep them in compressed format).

3.1.3 Image Format

Images are stored as an array of unsigned characters. Ideally, they would be in a structure, but the limitations of C prevents an efficient definition of a structure for an image while still allowing a compile-time declaration of the data for an image. The effective definition of the structure is as follows (with no padding between members):

```
typedef struct
{
    //
    // Specifies the format of the image data; will be one of
    // IMAGE_FMT_1BPP_UNCOMP, IMAGE_FMT_4BPP_UNCOMP, IMAGE_FMT_8BPP_UNCOMP,
    // IMAGE_FMT_1BPP_COMP, IMAGE_FMT_4BPP_COMP, or IMAGE_FMT_8BPP_COMP.  The
    // xxx_COMP varieties indicate that the image data is compressed.
    //
    unsigned char ucFormat;

    //
    // The width of the image in pixels.
    //
    unsigned short usWidth;

    //
    // The height of the image in pixels.
    //
    unsigned short usHeight;

    //
    // The image data.  This is the structure member that C can not handle
    // efficiently at compile time.
    //
    unsigned char pucData[];
}
tImage;
```

The format of the image data is dependent upon the number of bits per pixel and the use of compression. When compression is used, the uncompressed data will match the data that would appear in the image data for non-compressed images.

For 1 bit per pixel images, the image data is simply a sequence of bytes that describe the pixels in the image. Bits that are on represent pixels that should be drawn in the foreground color, and bits that are off represent pixels that should be drawn in the background color. The data is organized in row major order, with the first byte containing the left-most 8 pixels of the first scan line. The most significant bit of each byte corresponds to the left-most pixel, and padding bits are added at the end each scan line (if required) in order to ensure that each scan line starts at the beginning of a byte.

For 4 bits per pixel images, the first byte of the image data is the number of actual palette entries (which is actually stored as N - 1; in othter words, an 8 entry palette will have 7 in this byte. The next bytes are the palette for the image, which each entry being organized as a blue byte followed by a green byte followed by a red byte. Following the palette is the actual image data, where each nibble corresponds to an entry in the palette. The bytes are organized the same as for 1 bit per pixel images, and the most significant nibble of each byte corresponds to the left-most pixel.

For 8 bits per pixel images, the format is the same as 4 bits per pixel images with a larger palette and each byte containing exactly one pixel.

When the image data is compressed, the Lempel-Ziv-Storer-Szymanski algorithm is used to compress the data. This algorithm was originally published in the Journal of the ACM, 29(4):928-951, October 1982. The essence of this algorithm is to use the N most recent output bytes as a dictionary; the next encoded byte is either a literal byte (which is directly output) or a dictionary reference of up to M sequential bytes. For highly regular images (such as would be used for typical graphical controls), this works very well.

The compressed data is arranged as a sequence of 9 byte chunks. The first byte is a set of flags that indicate if the next 8 bytes are literal or dictionary references (one bit per byte). The most significant bit of the flags corresponds to the first byte. If the flag is clear, the byte is a literal; if it is set, the byte is a dictionary reference where the upper five bits are the dictionary offset and the lower three bits are the reference length (where 0 is 2 bytes, 1 is 3 bytes, and so on). Since a one byte dictionary reference takes one byte to encode, it is always stored as a literal so that length is able to range from 2 through 9 (providing a longer possible dictionary reference). The last 9 byte chunk may be truncated if all 8 data bytes are not required to encode the entire data sequence.

3.1.4 Font Format

Font data is stored with a scheme treats the rows of the font glyph as if they were connected side-by-side. Therefore, pixels from the end of one row can be combined with pixels from the beginning of the next row when storing. Fonts may be stored in an uncompressed format or may be compressed with a simple pixel-based RLE encoding. For either format, the format of the data for a font glyph is as follows:

- The first byte of the encoding is the number of bytes within the encoding (including the size byte).
- The second byte is the width of the character in pixels.
- The remaining bytes describe the pixels within the glyph.

For the uncompressed format, each 8-pixel quantity from the font glyph is stored as a byte in the glyph data. The most significant bit of each bit is the left-most pixel. So, for example, consider the following font glyph (for a non-existant character from a 14x8 font):

```
..............
..............
..............
......xx......
.....xxxx.....
...xxxxxxxx...
...xxxxxxxx...
..............
```

This would be stored as follows:

```
0x10, 0x0e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
0xc0, 0x07, 0x80, 0x7f, 0x81, 0xfe, 0x00, 0x00
```

Compression of the font data is via a per-glyph pixel RLE scheme. The encoded format of the font data is as follows:

- A non-zero byte encodes up to 15 consecutive off pixels followed by up to 15 consecutive on pixels. The upper nibble contains the number of off pixels and the lower nibble contains the number of on pixels. So, for example, `0x12` means that there is a single off pixel followed by two on pixels.
- A zero byte indicates repeated pixels. The next byte determines the size and type of the repeated pixels. The lower seven bits of the next byte specifies the number of bytes in the literal encoding (referred to as N). If the upper bit of the next byte is set, then there are N $*$ 8 on pixels. If the upper bit is not set, then there are N $*$ 8 off pixels.

The repeated pixel encoding is very useful for the several rows worth of off pixels that are present in many glyphs.

For the same glyph as above, the the compressed would be as follows, with an explanation of each byte or byte sequence:

- 0x0a: There are 10 bytes in the encoding of this glyph, including this byte.
- 0x0e: This glyph is 14 pixels wide.
- 0x00 0x06: The glyphs starts with 48 off pixels (6 bytes).
- 0x02: The fourth row has no addition off pixels before the two on pixels.
- 0xb4: The fourth row ends with 6 off pixels and the fifth row starts with 5 off pixels, making 11 off pixels. This is followed by 4 on pixels.
- 0x88: The fifth row ends with 5 off pixels, and the sixth row starts with 3 off pixels, making 8 off pixels. This is followed by 8 on pixels.
- 0x68: The sixth row ends with 3 off pixels, and the seventh row starts with 3 off pixels, making 6 off pixels. This is followed by 8 on pixels.
- 0xf0: The seventh row ends with 3 off pixels, and the eighth row has 14 off pixels. Since the maximum that can be encoded is 15, fifteen of the off pixels are here.
- 0x20: The remaining two off pixels, ending the glyph.

This results in a ten byte compressed font glyph, compared to the sixteen bytes required to describe the glyph without compression.

While being simplistic, this encoding method provides very effective results. Since the ASCII character set has a small number of strokes per character, the non-zero byte encoding format can effectively encode most occurrences of non-zero pixels and the repeated pixel encoding format can effectively encode the large run of zero pixels at the top and/or bottom of many glyphs.

3.2 Definitions

Data Structures

- tContext
- tDisplay
- tFont
- tRectangle

Defines

- DpyColorTranslate(pDisplay, ulValue)
- DpyFlush(pDisplay)
- DpyHeightGet(pDisplay)
- DpyLineDrawH(pDisplay, lX1, lX2, lY, ulValue)
- DpyLineDrawV(pDisplay, lX, lY1, lY2, ulValue)
- DpyPixelDraw(pDisplay, lX, lY, ulValue)
- DpyPixelDrawMultiple(pDisplay, lX, lY, lX0, lCount, lBPP, pucData,pucPalette)
- DpyRectFill(pDisplay, pRect, ulValue)
- DpyWidthGet(pDisplay)
- FONT_FMT_PIXEL_RLE
- FONT_FMT_UNCOMPRESSED
- GrContextBackgroundSet(pContext, ulValue)
- GrContextBackgroundSetTranslated(pContext, ulValue)
- GrContextDpyHeightGet(pContext)
- GrContextDpyWidthGet(pContext)
- GrContextFontSet(pContext, pFnt)
- GrContextForegroundSet(pContext, ulValue)
- GrContextForegroundSetTranslated(pContext, ulValue)
- GrFlush(pContext)
- GrFontBaselineGet(pFont)
- GrFontHeightGet(pFont)
- GrFontMaxWidthGet(pFont)
- GrImageColorsGet(pucImage)
- GrImageHeightGet(pucImage)
- GrImageWidthGet(pucImage)
- GrOffScreen1BPPSize(lWidth, lHeight)
- GrOffScreen4BPPSize(lWidth, lHeight)
- GrOffScreen8BPPSize(lWidth, lHeight)
- GrPixelDraw(pContext, lX, lY)
- GrStringBaselineGet(pContext)
- GrStringDrawCentered(pContext, pcString, lLength, lX, lY, bOpaque)
- GrStringHeightGet(pContext)
- GrStringMaxWidthGet(pContext)
- IMAGE_FMT_1BPP_COMP

- IMAGE_FMT_1BPP_UNCOMP
- IMAGE_FMT_4BPP_COMP
- IMAGE_FMT_4BPP_UNCOMP
- IMAGE_FMT_8BPP_COMP
- IMAGE_FMT_8BPP_UNCOMP

Functions

- void GrCircleDraw (const tContext ∗pContext, long lX, long lY, long lRadius)
- void GrCircleFill (const tContext ∗pContext, long lX, long lY, long lRadius)
- void GrContextClipRegionSet (tContext ∗pContext, tRectangle ∗pRect)
- void GrContextInit (tContext ∗pContext, const tDisplay ∗pDisplay)
- void GrImageDraw (const tContext ∗pContext, const unsigned char ∗pucImage, long lX, long lY)
- void GrLineDraw (const tContext ∗pContext, long lX1, long lY1, long lX2, long lY2)
- void GrLineDrawH (const tContext ∗pContext, long lX1, long lX2, long lY)
- void GrLineDrawV (const tContext ∗pContext, long lX, long lY1, long lY2)
- void GrOffScreen1BPPInit (tDisplay ∗pDisplay, unsigned char ∗pucImage, long lWidth, long lHeight)
- void GrOffScreen4BPPInit (tDisplay ∗pDisplay, unsigned char ∗pucImage, long lWidth, long lHeight)
- void GrOffScreen4BPPPaletteSet (tDisplay ∗pDisplay, unsigned long ∗pulPalette, unsigned long ulOffset, unsigned long ulCount)
- void GrOffScreen8BPPInit (tDisplay ∗pDisplay, unsigned char ∗pucImage, long lWidth, long lHeight)
- void GrOffScreen8BPPPaletteSet (tDisplay ∗pDisplay, unsigned long ∗pulPalette, unsigned long ulOffset, unsigned long ulCount)
- void GrRectDraw (const tContext ∗pContext, const tRectangle ∗pRect)
- void GrRectFill (const tContext ∗pContext, const tRectangle ∗pRect)
- void GrStringDraw (const tContext ∗pContext, const char ∗pcString, long lLength, long lX, long lY, unsigned long bOpaque)
- long GrStringWidthGet (const tContext ∗pContext, const char ∗pcString, long lLength)

3.2.1 Data Structure Documentation

3.2.1.1 tContext

Definition:

```
typedef struct
{
    long lSize;
    const tDisplay *pDisplay;
    tRectangle sClipRegion;
    unsigned long ulForeground;
    unsigned long ulBackground;
    const tFont *pFont;
}
tContext
```

Members:

lSize The size of this structure.

pDisplay The screen onto which drawing operations are performed.

sClipRegion The clipping region to be used when drawing onto the screen.

ulForeground The color used to draw primitives onto the screen.

ulBackground The background color used to draw primitives onto the screen.

pFont The font used to render text onto the screen.

Description:

This structure defines a drawing context to be used to draw onto the screen. Multiple drawing contexts may exist at any time.

3.2.1.2 tDisplay

Definition:

```
typedef struct
{
    long lSize;
    void *pvDisplayData;
    unsigned short usWidth;
    unsigned short usHeight;
    void (*pfnPixelDraw)(void *pvDisplayData,
                         long lX,
                         long lY,
                         unsigned long ulValue);
    void (*pfnPixelDrawMultiple)(void *pvDisplayData,
                                 long lX,
                                 long lY,
                                 long lX0,
                                 long lCount,
                                 long lBPP,
                                 const unsigned char *pucData,
                                 const unsigned char *pucPalette);
    void (*pfnLineDrawH)(void *pvDisplayData,
                         long lX1,
                         long lX2,
                         long lY,
                         unsigned long ulValue);
    void (*pfnLineDrawV)(void *pvDisplayData,
                         long lX,
                         long lY1,
                         long lY2,
                         unsigned long ulValue);
    void (*pfnRectFill)(void *pvDisplayData,
                        const tRectangle *pRect,
                        unsigned long ulValue);
    unsigned long (*pfnColorTranslate)(void *pvDisplayData,
                                       unsigned long ulValue);
    void (*pfnFlush)(void *pvDisplayData);
}
tDisplay
```

Members:

lSize The size of this structure.

pvDisplayData A pointer to display driver-specific data.

usWidth The width of this display.

usHeight The height of this display.

pfnPixelDraw A pointer to the function to draw a pixel on this display.

pfnPixelDrawMultiple A pointer to the function to draw multiple pixels on this display.

pfnLineDrawH A pointer to the function to draw a horizontal line on this display.

pfnLineDrawV A pointer to the function to draw a vertical line on this display.

pfnRectFill A pointer to the function to draw a filled rectangle on this display.

pfnColorTranslate A pointer to the function to translate 24-bit RGB colors to display-specific colors.

pfnFlush A pointer to the function to flush any cached drawing operations on this display.

Description:

This structure defines the characteristics of a display driver.

3.2.1.3 tFont

Definition:

```
typedef struct
{
    unsigned char ucFormat;
    unsigned char ucMaxWidth;
    unsigned char ucHeight;
    unsigned char ucBaseline;
    unsigned short pusOffset[96];
    const unsigned char *pucData;
}
tFont
```

Members:

ucFormat The format of the font. Can be one of FONT_FMT_UNCOMPRESSED or FONT_FMT_PIXEL_RLE.

ucMaxWidth The maximum width of a character; this is the width of the widest character in the font, though any individual character may be narrower than this width.

ucHeight The height of the character cell; this may be taller than the font data for the characters (to provide inter-line spacing).

ucBaseline The offset between the top of the character cell and the baseline of the glyph. The baseline is the bottom row of a capital letter, below which only the descenders of the lower case letters occur.

pusOffset The offset within pucData to the data for each character in the font.

pucData A pointer to the data for the font.

Description:

This structure describes a font used for drawing text onto the screen.

3.2.1.4 tRectangle

Definition:

```
typedef struct
{
    short sXMin;
    short sYMin;
    short sXMax;
    short sYMax;
}
tRectangle
```

Members:

sXMin The minimum X coordinate of the rectangle.

sYMin The minimum Y coordinate of the rectangle.

sXMax The maximum X coordinate of the rectangle.

sYMax The maximum Y coordinate of the rectangle.

Description:

This structure defines the extents of a rectangle. All points greater than or equal to the minimum and less than or equal to the maximum are part of the rectangle.

3.2.2 Define Documentation

3.2.2.1 DpyColorTranslate

Translates a 24-bit RGB color to a display driver-specific color.

Definition:

```
#define DpyColorTranslate(pDisplay,
                          ulValue)
```

Parameters:

pDisplay is the pointer to the display driver structure for the display to operate upon.

ulValue is the 24-bit RGB color. The least-significant byte is the blue channel, the next byte is the green channel, and the third byte is the red channel.

Description:

This function translates a 24-bit RGB color into a value that can be written into the display's frame buffer in order to reproduce that color, or the closest possible approximation of that color.

Returns:

Returns the display-driver specific color.

3.2.2.2 DpyFlush

Flushes cached drawing operations.

Definition:

```
#define DpyFlush(pDisplay)
```

Parameters:

pDisplay is the pointer to the display driver structure for the display to operate upon.

Description:

This function flushes any cached drawing operations on a display.

Returns:

None.

3.2.2.3 DpyHeightGet

Gets the height of the display.

Definition:

```
#define DpyHeightGet(pDisplay)
```

Parameters:

pDisplay is a pointer to the display driver structure for the display to query.

Description:

This function determines the height of the display.

Returns:

Returns the height of the display in pixels.

3.2.2.4 DpyLineDrawH

Draws a horizontal line on a display.

Definition:

```
#define DpyLineDrawH(pDisplay,
                     lX1,
                     lX2,
                     lY,
                     ulValue)
```

Parameters:

pDisplay is the pointer to the display driver structure for the display to operate upon.

lX1 is the starting X coordinate of the line.

lX2 is the ending X coordinate of the line.

lY is the Y coordinate of the line.

ulValue is the color to draw the line.

Description:

This function draws a horizontal line on a display. This assumes that clipping has already been performed, and that both end points of the line are within the extents of the display.

Returns:

None.

3.2.2.5 DpyLineDrawV

Draws a vertical line on a display.

Definition:
```
#define DpyLineDrawV(pDisplay,
                     lX,
                     lY1,
                     lY2,
                     ulValue)
```

Parameters:

pDisplay is the pointer to the display driver structure for the display to operate upon.

lX is the X coordinate of the line.

lY1 is the starting Y coordinate of the line.

lY2 is the ending Y coordinate of the line.

ulValue is the color to draw the line.

Description:

This function draws a vertical line on a display. This assumes that clipping has already been performed, and that both end points of the line are within the extents of the display.

Returns:

None.

3.2.2.6 DpyPixelDraw

Draws a pixel on a display.

Definition:
```
#define DpyPixelDraw(pDisplay,
                     lX,
                     lY,
                     ulValue)
```

Parameters:

pDisplay is the pointer to the display driver structure for the display to operate upon.

lX is the X coordinate of the pixel.

lY is the Y coordinate of the pixel.

ulValue is the color to draw the pixel.

Description:

This function draws a pixel on a display. This assumes that clipping has already been performed.

Returns:

None.

3.2.2.7 DpyPixelDrawMultiple

Draws a horizontal sequence of pixels on a display.

Definition:

```
#define DpyPixelDrawMultiple(pDisplay,
                             lX,
                             lY,
                             lX0,
                             lCount,
                             lBPP,
                             pucData,
                             pucPalette)
```

Parameters:

pDisplay is the pointer to the display driver structure for the display to operate upon.

lX is the X coordinate of the first pixel.

lY is the Y coordinate of the first pixel.

lX0 is sub-pixel offset within the pixel data, which is valid for 1 or 4 bit per pixel formats.

lCount is the number of pixels to draw.

lBPP is the number of bits per pixel; must be 1, 4, or 8.

pucData is a pointer to the pixel data. For 1 and 4 bit per pixel formats, the most significant bit(s) represent the left-most pixel.

pucPalette is a pointer to the palette used to draw the pixels.

Description:

This function draws a horizontal sequence of pixels on a display, using the supplied palette. For 1 bit per pixel format, the palette contains pre-translated colors; for 4 and 8 bit per pixel formats, the palette contains 24-bit RGB values that must be translated before being written to the display.

Returns:

None.

3.2.2.8 DpyRectFill

Fills a rectangle on a display.

Definition:

```
#define DpyRectFill(pDisplay,
                    pRect,
                    ulValue)
```

Parameters:

pDisplay is the pointer to the display driver structure for the display to operate upon.

pRect is a pointer to the structure describing the rectangle to fill.

ulValue is the color to fill the rectangle.

Description:

This function fills a rectangle on the display. This assumes that clipping has already been performed, and that all sides of the rectangle are within the extents of the display.

Returns:
None.

3.2.2.9 DpyWidthGet

Gets the width of the display.

Definition:
```
#define DpyWidthGet(pDisplay)
```

Parameters:
pDisplay is a pointer to the display driver structure for the display to query.

Description:
This function determines the width of the display.

Returns:
Returns the width of the display in pixels.

3.2.2.10 FONT_FMT_PIXEL_RLE

Definition:
```
#define FONT_FMT_PIXEL_RLE
```

Description:
Indicates that the font data is stored using a pixel-based RLE format.

3.2.2.11 FONT_FMT_UNCOMPRESSED

Definition:
```
#define FONT_FMT_UNCOMPRESSED
```

Description:
Indicates that the font data is stored in an uncompressed format.

3.2.2.12 GrContextBackgroundSet

Sets the background color to be used.

Definition:
```
#define GrContextBackgroundSet(pContext,
                               ulValue)
```

Parameters:
pContext is a pointer to the drawing context to modify.
ulValue is the 24-bit RGB color to be used.

Description:
This function sets the background color to be used for drawing operations in the specified drawing context.

Returns:
None.

3.2.2.13 GrContextBackgroundSetTranslated

Sets the background color to be used.

Definition:
```
#define GrContextBackgroundSetTranslated(pContext,
                                         ulValue)
```

Parameters:
pContext is a pointer to the drawing context to modify.
ulValue is the display driver-specific color to be used.

Description:
This function sets the background color to be used for drawing operations in the specified drawing context, using a color that has been previously translated to a driver-specific color (for example, via DpyColorTranslate()).

Returns:
None.

3.2.2.14 GrContextDpyHeightGet

Gets the height of the display being used by this drawing context.

Definition:
```
#define GrContextDpyHeightGet(pContext)
```

Parameters:
pContext is a pointer to the drawing context to query.

Description:
This function returns the height of the display that is being used by this drawing context.

Returns:
Returns the height of the display in pixels.

3.2.2.15 GrContextDpyWidthGet

Gets the width of the display being used by this drawing context.

Definition:
```
#define GrContextDpyWidthGet(pContext)
```

Parameters:
pContext is a pointer to the drawing context to query.

Description:
This function returns the width of the display that is being used by this drawing context.

Returns:
Returns the width of the display in pixels.

3.2.2.16 GrContextFontSet

Sets the font to be used.

Definition:
```
#define GrContextFontSet(pContext,
                         pFnt)
```

Parameters:
pContext is a pointer to the drawing context to modify.
pFnt is a pointer to the font to be used.

Description:
This function sets the font to be used for string drawing operations in the specified drawing context.

Returns:
None.

3.2.2.17 GrContextForegroundSet

Sets the foreground color to be used.

Definition:
```
#define GrContextForegroundSet(pContext,
                               ulValue)
```

Parameters:
pContext is a pointer to the drawing context to modify.
ulValue is the 24-bit RGB color to be used.

Description:
This function sets the color to be used for drawing operations in the specified drawing context.

Returns:
None.

3.2.2.18 GrContextForegroundSetTranslated

Sets the foreground color to be used.

Definition:

```
#define GrContextForegroundSetTranslated(pContext,
                                         ulValue)
```

Parameters:

pContext is a pointer to the drawing context to modify.

ulValue is the display driver-specific color to be used.

Description:

This function sets the foreground color to be used for drawing operations in the specified drawing context, using a color that has been previously translated to a driver-specific color (for example, via DpyColorTranslate()).

Returns:

None.

3.2.2.19 GrFlush

Flushes any cached drawing operations.

Definition:

```
#define GrFlush(pContext)
```

Parameters:

pContext is a pointer to the drawing context to use.

Description:

This function flushes any cached drawing operations. For display drivers that draw into a local frame buffer before writing to the actual display, calling this function will cause the display to be updated to match the contents of the local frame buffer.

Returns:

None.

3.2.2.20 GrFontBaselineGet

Gets the baseline of a font.

Definition:

```
#define GrFontBaselineGet(pFont)
```

Parameters:

pFont is a pointer to the font to query.

Description:

This function determines the baseline position of a font. The baseline is the offset between the top of the font and the bottom of the capital letters. The only font data that exists below the baseline are the descenders on some lower-case letters (such as “y”).

Returns:
Returns the baseline of the font, in pixels.

3.2.2.21 GrFontHeightGet

Gets the height of a font.

Definition:
```
#define GrFontHeightGet(pFont)
```

Parameters:
pFont is a pointer to the font to query.

Description:
This function determines the height of a font. The height is the offset between the top of the font and the bottom of the font, including any ascenders and descenders.

Returns:
Returns the height of the font, in pixels.

3.2.2.22 GrFontMaxWidthGet

Gets the maximum width of a font.

Definition:
```
#define GrFontMaxWidthGet(pFont)
```

Parameters:
pFont is a pointer to the font to query.

Description:
This function determines the maximum width of a font. The maximum width is the width of the widest individual character in the font.

Returns:
Returns the maximum width of the font, in pixels.

3.2.2.23 GrImageColorsGet

Gets the number of colors in an image.

Definition:
```
#define GrImageColorsGet(pucImage)
```

Parameters:
pucImage is a pointer to the image to query.

Description:
This function determines the number of colors in the palette of an image. This is only valid for 4bpp and 8bpp images; 1bpp images do not contain a palette.

Returns:
Returns the number of colors in the image.

3.2.2.24 GrImageHeightGet

Gets the height of an image.

Definition:
```
#define GrImageHeightGet(pucImage)
```

Parameters:
pucImage is a pointer to the image to query.

Description:
This function determines the height of an image in pixels.

Returns:
Returns the height of the image in pixels.

3.2.2.25 GrImageWidthGet

Gets the width of an image.

Definition:
```
#define GrImageWidthGet(pucImage)
```

Parameters:
pucImage is a pointer to the image to query.

Description:
This function determines the width of an image in pixels.

Returns:
Returns the width of the image in pixels.

3.2.2.26 GrOffScreen1BPPSize

Determines the size of the buffer for a 1 BPP off-screen image.

Definition:
```
#define GrOffScreen1BPPSize(lWidth,
                            lHeight)
```

Parameters:
lWidth is the width of the image in pixels.
lHeight is the height of the image in pixels.

Description:
This function determines the size of the memory buffer required to hold a 1 BPP off-screen image of the specified geometry.

Returns:
Returns the number of bytes required by the image.

3.2.2.27 GrOffScreen4BPPSize

Determines the size of the buffer for a 4 BPP off-screen image.

Definition:
```
#define GrOffScreen4BPPSize(lWidth,
                            lHeight)
```

Parameters:
lWidth is the width of the image in pixels.
lHeight is the height of the image in pixels.

Description:
This function determines the size of the memory buffer required to hold a 4 BPP off-screen image of the specified geometry.

Returns:
Returns the number of bytes required by the image.

3.2.2.28 GrOffScreen8BPPSize

Determines the size of the buffer for an 8 BPP off-screen image.

Definition:
```
#define GrOffScreen8BPPSize(lWidth,
                            lHeight)
```

Parameters:
lWidth is the width of the image in pixels.
lHeight is the height of the image in pixels.

Description:
This function determines the size of the memory buffer required to hold an 8 BPP off-screen image of the specified geometry.

Returns:
Returns the number of bytes required by the image.

3.2.2.29 GrPixelDraw

Draws a pixel.

Definition:
```
#define GrPixelDraw(pContext,
                    lX,
                    lY)
```

Parameters:

pContext is a pointer to the drawing context to use.

lX is the X coordinate of the pixel.

lY is the Y coordinate of the pixel.

Description:

This function draws a pixel if it resides within the clipping region.

Returns:

None.

3.2.2.30 GrStringBaselineGet

Gets the baseline of a string.

Definition:

```
#define GrStringBaselineGet(pContext)
```

Parameters:

pContext is a pointer to the drawing context to query.

Description:

This function determines the baseline position of a string. The baseline is the offset between the top of the string and the bottom of the capital letters. The only string data that exists below the baseline are the descenders on some lower-case letters (such as “y”).

Returns:

Returns the baseline of the string, in pixels.

3.2.2.31 GrStringDrawCentered

Draws a centered string.

Definition:

```
#define GrStringDrawCentered(pContext,
                             pcString,
                             lLength,
                             lX,
                             lY,
                             bOpaque)
```

Parameters:

pContext is a pointer to the drawing context to use.

pcString is a pointer to the string to be drawn.

lLength is the number of characters from the string that should be drawn on the screen.

lX is the X coordinate of the center of the string position on the screen.

lY is the Y coordinate of the center of the string position on the screen.

bOpaque is **true** if the background of each character should be drawn and **false** if it should not (leaving the background as is).

Description:

This function draws a string of test on the screen centered upon the provided position. The *lLength* parameter allows a portion of the string to be examined without having to insert a NULL character at the stopping point (which would not be possible if the string was located in flash); specifying a length of -1 will cause the entire string to be rendered (subject to clipping).

Returns:

None.

3.2.2.32 GrStringHeightGet

Gets the height of a string.

Definition:

```
#define GrStringHeightGet(pContext)
```

Parameters:

pContext is a pointer to the drawing context to query.

Description:

This function determines the height of a string. The height is the offset between the top of the string and the bottom of the string, including any ascenders and descenders. Note that this will not account for the case where the string in question does not have any characters that use descenders but the font in the drawing context does contain characters with descenders.

Returns:

Returns the height of the string, in pixels.

3.2.2.33 GrStringMaxWidthGet

Gets the maximum width of a character in a string.

Definition:

```
#define GrStringMaxWidthGet(pContext)
```

Parameters:

pContext is a pointer to the drawing context to query.

Description:

This function determines the maximum width of a character in a string. The maximum width is the width of the widest individual character in the font used to render the string, which may be wider than the widest character that is used to render a particular string.

Returns:

Returns the maximum width of a character in a string, in pixels.

3.2.2.34 IMAGE_FMT_1BPP_COMP

Definition:

```
#define IMAGE_FMT_1BPP_COMP
```

Description:

Indicates that the image data is compressed and represents each pixel with a single bit.

3.2.2.35 IMAGE_FMT_1BPP_UNCOMP

Definition:

```
#define IMAGE_FMT_1BPP_UNCOMP
```

Description:

Indicates that the image data is not compressed and represents each pixel with a single bit.

3.2.2.36 IMAGE_FMT_4BPP_COMP

Definition:

```
#define IMAGE_FMT_4BPP_COMP
```

Description:

Indicates that the image data is compressed and represents each pixel with four bits.

3.2.2.37 IMAGE_FMT_4BPP_UNCOMP

Definition:

```
#define IMAGE_FMT_4BPP_UNCOMP
```

Description:

Indicates that the image data is not compressed and represents each pixel with four bits.

3.2.2.38 IMAGE_FMT_8BPP_COMP

Definition:

```
#define IMAGE_FMT_8BPP_COMP
```

Description:

Indicates that the image data is compressed and represents each pixel with eight bits.

3.2.2.39 IMAGE_FMT_8BPP_UNCOMP

Definition:

```
#define IMAGE_FMT_8BPP_UNCOMP
```

Description:

Indicates that the image data is not compressed and represents each pixel with eight bits.

3.2.3 Function Documentation

3.2.3.1 GrCircleDraw

Draws a circle.

Prototype:

```
void
GrCircleDraw(const tContext *pContext,
             long lX,
             long lY,
             long lRadius)
```

Parameters:

pContext is a pointer to the drawing context to use.

lX is the X coordinate of the center of the circle.

lY is the Y coordinate of the center of the circle.

lRadius is the radius of the circle.

Description:

This function draws a circle, utilizing the Bresenham circle drawing algorithm. The extent of the circle is from *lX* - *lRadius* to *lX* + *lRadius* and *lY* - *lRadius* to *lY* + *lRadius*, inclusive.

Returns:

None.

3.2.3.2 GrCircleFill

Draws a filled circle.

Prototype:

```
void
GrCircleFill(const tContext *pContext,
             long lX,
             long lY,
             long lRadius)
```

Parameters:

pContext is a pointer to the drawing context to use.

lX is the X coordinate of the center of the circle.

lY is the Y coordinate of the center of the circle.

lRadius is the radius of the circle.

Description:

This function draws a filled circle, utilizing the Bresenham circle drawing algorithm. The extent of the circle is from *lX* - *lRadius* to *lX* + *lRadius* and *lY* - *lRadius* to *lY* + *lRadius*, inclusive.

Returns:

None.

3.2.3.3 GrContextClipRegionSet

Sets the extents of the clipping region.

Prototype:

```
void
GrContextClipRegionSet(tContext *pContext,
                       tRectangle *pRect)
```

Parameters:

pContext is a pointer to the drawing context to use.

pRect is a pointer to the structure containing the extents of the clipping region.

Description:

This function sets the extents of the clipping region. The clipping region is not allowed to exceed the extents of the screen, but may be a portion of the screen.

The supplied coordinate are inclusive; *sXMin* of 1 and *sXMax* of 1 will define a clipping region that will display only the pixels in the X = 1 column. A consequence of this is that the clipping region must contain at least one row and one column.

Returns:

None.

3.2.3.4 GrContextInit

Initializes a drawing context.

Prototype:

```
void
GrContextInit(tContext *pContext,
              const tDisplay *pDisplay)
```

Parameters:

pContext is a pointer to the drawing context to initialize.

pDisplay is a pointer to the tDisplayInfo structure that describes the display driver to use.

Description:

This function initializes a drawing context, preparing it for use. The provided display driver will be used for all subsequent graphics operations, and the default clipping region will be set to the extent of the screen.

Returns:

None.

3.2.3.5 GrImageDraw

Draws a bitmap image.

Prototype:

```
void
GrImageDraw(const tContext *pContext,
            const unsigned char *pucImage,
            long lX,
            long lY)
```

Parameters:

pContext is a pointer to the drawing context to use.

pucImage is a pointer to the image to draw.

lX is the X coordinate of the upper left corner of the image.

lY is the Y coordinate of the upper left corner of the image.

Description:

This function draws a bitmap image. The image may be 1 bit per pixel (using the foreground and background color from the drawing context), 4 bits per pixel (using a palette supplied in the image data), or 8 bits per pixel (using a palette supplied in the image data). It can be uncompressed data, or it can be compressed using the Lempel-Ziv-Storer-Szymanski algorithm (as published in the Journal of the ACM, 29(4):928-951, October 1982).

Returns:

None.

3.2.3.6 GrLineDraw

Draws a line.

Prototype:

```
void
GrLineDraw(const tContext *pContext,
           long lX1,
           long lY1,
           long lX2,
           long lY2)
```

Parameters:

pContext is a pointer to the drawing context to use.

lX1 is the X coordinate of the start of the line.

lY1 is the Y coordinate of the start of the line.

lX2 is the X coordinate of the end of the line.

lY2 is the Y coordinate of the end of the line.

Description:

This function draws a line, utilizing GrLineDrawH() and GrLineDrawV() to draw the line as efficiently as possible. The line is clipped to the clippping rectangle using the Cohen-Sutherland clipping algorithm, and then scan converted using Bresenham's line drawing algorithm.

Returns:

None.

3.2.3.7 GrLineDrawH

Draws a horizontal line.

Prototype:

```
void
GrLineDrawH(const tContext *pContext,
            long lX1,
            long lX2,
            long lY)
```

Parameters:

pContext is a pointer to the drawing context to use.

lX1 is the X coordinate of one end of the line.

lX2 is the X coordinate of the other end of the line.

lY is the Y coordinate of the line.

Description:

This function draws a horizontal line, taking advantage of the fact that the line is horizontal to draw it more efficiently. The clipping of the horizontal line to the clipping rectangle is performed within this routine; the display driver's horizontal line routine is used to perform the actual line drawing.

Returns:

None.

3.2.3.8 GrLineDrawV

Draws a vertical line.

Prototype:

```
void
GrLineDrawV(const tContext *pContext,
            long lX,
            long lY1,
            long lY2)
```

Parameters:

pContext is a pointer to the drawing context to use.

lX is the X coordinate of the line.

lY1 is the Y coordinate of one end of the line.

lY2 is the Y coordinate of the other end of the line.

Description:

This function draws a vertical line, taking advantage of the fact that the line is vertical to draw it more efficiently. The clipping of the vertical line to the clipping rectangle is performed within this routine; the display driver's vertical line routine is used to perform the actual line drawing.

Returns:

None.

3.2.3.9 GrOffScreen1BPPInit

Initializes a 1 BPP off-screen buffer.

Prototype:

```
void
GrOffScreen1BPPInit(tDisplay *pDisplay,
                    unsigned char *pucImage,
                    long lWidth,
                    long lHeight)
```

Parameters:

pDisplay is a pointer to the display structure to be configured for the 1 BPP off-screen buffer.

pucImage is a pointer to the image buffer to be used for the off-screen buffer.

lWidth is the width of the image buffer in pixels.

lHeight is the height of the image buffer in pixels.

Description:

This function initializes a display structure, preparing it to draw into the supplied image buffer. The image buffer is assumed to be large enough to hold an image of the specified geometry.

Returns:

None.

3.2.3.10 GrOffScreen4BPPInit

Initializes a 4 BPP off-screen buffer.

Prototype:

```
void
GrOffScreen4BPPInit(tDisplay *pDisplay,
                    unsigned char *pucImage,
                    long lWidth,
                    long lHeight)
```

Parameters:

pDisplay is a pointer to the display structure to be configured for the 4 BPP off-screen buffer.

pucImage is a pointer to the image buffer to be used for the off-screen buffer.

lWidth is the width of the image buffer in pixels.

lHeight is the height of the image buffer in pixels.

Description:

This function initializes a display structure, preparing it to draw into the supplied image buffer. The image buffer is assumed to be large enough to hold an image of the specified geometry.

Returns:

None.

3.2.3.11 GrOffScreen4BPPPaletteSet

Sets the palette of a 4 BPP off-screen buffer.

Prototype:

```
void
GrOffScreen4BPPPaletteSet(tDisplay *pDisplay,
                          unsigned long *pulPalette,
                          unsigned long ulOffset,
                          unsigned long ulCount)
```

Parameters:

pDisplay is a pointer to the display structure for the 4 BPP off-screen buffer.

pulPalette is a pointer to the array of 24-bit RGB values to be placed into the palette.

ulOffset is the starting offset into the image palette.

ulCount is the number of palette entries to set.

Description:

This function sets the entries of the palette used by the 4 BPP off-screen buffer. The palette is used to select colors for drawing via GrOffScreen4BPPColorTranslate(), and for the final rendering of the image to a real display via GrImageDraw().

Returns:

None.

3.2.3.12 GrOffScreen8BPPInit

Initializes an 8 BPP off-screen buffer.

Prototype:

```
void
GrOffScreen8BPPInit(tDisplay *pDisplay,
                    unsigned char *pucImage,
                    long lWidth,
                    long lHeight)
```

Parameters:

pDisplay is a pointer to the display structure to be configured for the 4 BPP off-screen buffer.

pucImage is a pointer to the image buffer to be used for the off-screen buffer.

lWidth is the width of the image buffer in pixels.

lHeight is the height of the image buffer in pixels.

Description:

This function initializes a display structure, preparing it to draw into the supplied image buffer. The image buffer is assumed to be large enough to hold an image of the specified geometry.

Returns:

None.

3.2.3.13 GrOffScreen8BPPPaletteSet

Sets the palette of an 8 BPP off-screen buffer.

Prototype:

```
void
GrOffScreen8BPPPaletteSet(tDisplay *pDisplay,
                          unsigned long *pulPalette,
                          unsigned long ulOffset,
                          unsigned long ulCount)
```

Parameters:

pDisplay is a pointer to the display structure for the 4 BPP off-screen buffer.

pulPalette is a pointer to the array of 24-bit RGB values to be placed into the palette.

ulOffset is the starting offset into the image palette.

ulCount is the number of palette entries to set.

Description:

This function sets the entries of the palette used by the 8 BPP off-screen buffer. The palette is used to select colors for drawing via GrOffScreen4BPPColorTranslate(), and for the final rendering of the image to a real display via GrImageDraw().

Returns:

None.

3.2.3.14 GrRectDraw

Draws a rectangle.

Prototype:

```
void
GrRectDraw(const tContext *pContext,
           const tRectangle *pRect)
```

Parameters:

pContext is a pointer to the drawing context to use.

pRect is a pointer to the structure containing the extents of the rectangle.

Description:

This function draws a rectangle. The rectangle will extend from *lXMin* to *lXMax* and *lYMin* to *lYMax*, inclusive.

Returns:

None.

3.2.3.15 GrRectFill

Draws a filled rectangle.

Prototype:

```
void
GrRectFill(const tContext *pContext,
           const tRectangle *pRect)
```

Parameters:

pContext is a pointer to the drawing context to use.

pRect is a pointer to the structure containing the extents of the rectangle.

Description:

This function draws a filled rectangle. The rectangle will extend from *lXMin* to *lXMax* and *lYMin* to *lYMax*, inclusive. The clipping of the rectangle to the clipping rectangle is performed within this routine; the display driver's rectangle fill routine is used to perform the actual rectangle fill.

Returns:

None.

3.2.3.16 GrStringDraw

Draws a string.

Prototype:

```
void
GrStringDraw(const tContext *pContext,
             const char *pcString,
             long lLength,
             long lX,
             long lY,
             unsigned long bOpaque)
```

Parameters:

pContext is a pointer to the drawing context to use.

pcString is a pointer to the string to be drawn.

lLength is the number of characters from the string that should be drawn on the screen.

lX is the X coordinate of the upper left corner of the string position on the screen.

lY is the Y coordinate of the upper left corner of the string position on the screen.

bOpaque is true of the background of each character should be drawn and false if it should not (leaving the background as is).

Description:

This function draws a string of test on the screen. The *lLength* parameter allows a portion of the string to be examined without having to insert a NULL character at the stopping point (which would not be possible if the string was located in flash); specifying a length of -1 will cause the entire string to be rendered (subject to clipping).

Returns:

None.

3.2.3.17 GrStringWidthGet

Determines the width of a string.

Prototype:

```
long
GrStringWidthGet(const tContext *pContext,
                 const char *pcString,
                 long lLength)
```

Parameters:

pContext is a pointer to the drawing context to use.

pcString is the string in question.

lLength is the length of the string.

Description:

This function determines the width of a string (or portion of the string) when drawn with a particular font. The *lLength* parameter allows a portion of the string to be examined without having to insert a NULL character at the stopping point (would not be possible if the string was located in flash); specifying a length of -1 will cause the width of the entire string to be computed.

Returns:

Returns the width of the string in pixels.

4 Widget Framework

4.1 Introduction

A widget is an entity that ties together the rendering of a graphical element on the screen with the response to input from the user. An example of a widget is a button that performs an application-defined action when it is pressed.

The widget framework provides a generic means of dealing with a wide variety of widgets. Each widget has a message handler that responds to a set of generic messages; for example, the WIDGET_MSG_PAINT message is sent to request that the widget draw itself onto the screen.

The widgets are organized in a tree structure, and can be dynamically added or removed from the active widget tree. The tree structure allows messages to be delivered in a controlled manner. For example, the WIDGET_MSG_PAINT message is delivered to a widget's parent before it is delivered to that widget (so that the child is not obscured by its enclosing parent). Each message is delivered in either top-down or bottom-up order based on the semantics of the message.

Widgets can be created at run-time by calling functions or at compile-time by using global structure definitions. Helper macros are provided by the individual widgets for defining the global structures.

4.2 Definitions

Data Structures

- tWidget

Defines

- WIDGET_MSG_PAINT
- WIDGET_MSG_PTR_DOWN
- WIDGET_MSG_PTR_MOVE
- WIDGET_MSG_PTR_UP
- WIDGET_ROOT
- WidgetPaint(pWidget)

Functions

- void WidgetAdd (tWidget ∗pParent, tWidget ∗pWidget)
- long WidgetDefaultMsgProc (tWidget ∗pWidget, unsigned long ulMessage, unsigned long ulParam1, unsigned long ulParam2)

- long WidgetMessageQueueAdd (tWidget ∗pWidget, unsigned long ulMessage, unsigned long ulParam1, unsigned long ulParam2, unsigned long bPostOrder, unsigned long bStopOnSuccess)
- void WidgetMessageQueueProcess (void)
- unsigned long WidgetMessageSendPostOrder (tWidget ∗pWidget, unsigned long ulMessage, unsigned long ulParam1, unsigned long ulParam2, unsigned long bStopOnSuccess)
- unsigned long WidgetMessageSendPreOrder (tWidget ∗pWidget, unsigned long ulMessage, unsigned long ulParam1, unsigned long ulParam2, unsigned long bStopOnSuccess)
- long WidgetPointerMessage (unsigned long ulMessage, long lX, long lY)
- void WidgetRemove (tWidget ∗pWidget)

4.2.1 Data Structure Documentation

4.2.1.1 tWidget

Definition:

```
typedef struct
{
    long lSize;
    tWidget *pParent;
    tWidget *pNext;
    tWidget *pChild;
    const tDisplay *pDisplay;
    tRectangle sPosition;
    long (*pfnMsgProc)(tWidget *pWidget,
                       unsigned long ulMessage,
                       unsigned long ulParam1,
                       unsigned long ulParam2);
}
tWidget
```

Members:

lSize The size of this structure. This will be the size of the full structure, not just the generic widget subset.

pParent A pointer to this widget's parent widget.

pNext A pointer to this widget's first sibling widget.

pChild A pointer to this widget's first child widget.

pDisplay A pointer to the display on which this widget resides.

sPosition The rectangle that encloses this widget.

pfnMsgProc The procedure that handles messages sent to this widget.

Description:

The structure that describes a generic widget. This structure is the base “class” for all other widgets.

4.2.2 Define Documentation

4.2.2.1 WIDGET_MSG_PAINT

Definition:

```
#define WIDGET_MSG_PAINT
```

Description:

This message is sent to indicate that the widget should draw itself on the display. Neither *ulParam1* nor *ulParam2* are used by this message. This message is delivered in top-down order.

4.2.2.2 WIDGET_MSG_PTR_DOWN

Definition:

```
#define WIDGET_MSG_PTR_DOWN
```

Description:

This message is sent to indicate that the pointer is now down. *ulParam1* is the X coordinate of the location where the pointer down event occurred, and *ulParam2* is the Y coordinate. This message is delivered in bottom-up order.

4.2.2.3 WIDGET_MSG_PTR_MOVE

Definition:

```
#define WIDGET_MSG_PTR_MOVE
```

Description:

This message is sent to indicate that the pointer has moved while being down. *ulParam1* is the X coordinate of the new pointer location, and *ulParam2* is the Y coordinate. This message is delivered in bottom-up order.

4.2.2.4 WIDGET_MSG_PTR_UP

Definition:

```
#define WIDGET_MSG_PTR_UP
```

Description:

This message is sent to indicate that the pointer is now up. *ulParam1* is the X coordinate of the location where the pointer up event occurred, and *ulParam2* is the Y coordinate. This message is delivered in bottom-up order.

4.2.2.5 WIDGET_ROOT

Definition:

```
#define WIDGET_ROOT
```

Description:

The widget at the root of the widget tree. This can be used when constructing a widget tree at compile time (used as the pParent argument to a widget declaration) or as the pWidget argument to an API (such as WidgetPaint() to paint the entire widget tree).

4.2.2.6 WidgetPaint

Requests a redraw of the widget tree.

Definition:

```
#define WidgetPaint(pWidget)
```

Parameters:

pWidget is a pointer to the widget tree to paint.

Description:

This function sends a **WIDGET_MSG_PAINT** message to the given widgets, and all of the widget beneath it, so that they will draw or redraw themselves on the display. The actual drawing will occur when this message is retrieved from the message queue and processed.

Returns:

Returns 1 if the message was added to the message queue and 0 if it cound not be added (due to the queue being full).

4.2.3 Function Documentation

4.2.3.1 WidgetAdd

Adds a widget to the widget tree.

Prototype:

```
void
WidgetAdd(tWidget *pParent,
          tWidget *pWidget)
```

Parameters:

pParent is the parent for the widget; if this is zero then the root of the widget tree will be used.

pWidget is the widget to add.

Description:

This function adds a widget to the widget tree at the given position within the tree. The widget will become the last child of its parent, and will therefore be searched after the existing children.

The added widget can be a full widget tree, allowing addition of an entire heirarchy all at once (for example, adding an entire screen to the widget tree all at once). In this case, it is the responsibility of the caller to ensure that the pParent field of each widget in the added tree is correctly set (in other words, only the widget pointed to by *pWidget* is updated to properly reside in the tree).

It is the responsibility of the caller to initialize the pNext and pChild field of the added widget; either of these fields being non-zero results in a pre-defined tree of widgets being added instead of a single one.

Returns:
None.

4.2.3.2 WidgetDefaultMsgProc

Handles widget messages.

Prototype:
```
long
WidgetDefaultMsgProc(tWidget *pWidget,
                     unsigned long ulMessage,
                     unsigned long ulParam1,
                     unsigned long ulParam2)
```

Parameters:
pWidget is a pointer to the widget.
ulMessage is the message to be processed.
ulParam1 is the first parameter to the message.
ulParam2 is the second parameter to the message.

Description:
This function is a default handler for widget messages; it simply ignores all messages sent to it. This is used as the message handler for the root widget, and should be called by the message handler for other widgets when they do not explicitly handle the provided message (in case new messages are added that require some default but override-able processing).

Returns:
Always returns 0.

4.2.3.3 WidgetMessageQueueAdd

Adds a message to the widget message queue.

Prototype:
```
long
WidgetMessageQueueAdd(tWidget *pWidget,
                      unsigned long ulMessage,
                      unsigned long ulParam1,
                      unsigned long ulParam2,
                      unsigned long bPostOrder,
                      unsigned long bStopOnSuccess)
```

Parameters:
pWidget is the widget to which the message should be sent.
ulMessage is the message to be sent.
ulParam1 is the first parameter to the message.
ulParam2 is the second parameter to the message.
bPostOrder is **true** if the message should be sent via a post-order search, and **false** if it should be sent via a pre-order search.

bStopOnSuccess is **true** if the message should be sent to widgets until one returns success, and **false** if it should be sent to all widgets.

Description:

This function places a widget message into the message queue for later processing. The messages are removed from the queue by WidgetMessageQueueProcess() and sent to the appropriate place.

It is safe for code which interrupts WidgetMessageQueueProcess() (or called by it) to call this function to send a message. It is not safe for code which interrupts this function to call this function as well; it is up to the caller to guarantee that the later sequence never occurs.

Returns:

Returns 1 if the message was added to the queue, and 0 if it could not be added since the queue is full.

4.2.3.4 WidgetMessageQueueProcess

Processes the messages in the widget message queue.

Prototype:

```
void
WidgetMessageQueueProcess(void)
```

Description:

This function extracts messages from the widget message queue one at a time and processes them. If the processing of a widget message requires that a new message be sent, it is acceptable to call WidgetMessageQueueAdd(). It is also acceptable for code which interrupts this function to call WidgetMessageQueueAdd() to send more messages. In both cases, the newly added message will also be processed before this function returns.

Returns:

None.

4.2.3.5 WidgetMessageSendPostOrder

Sends a message to a widget tree via a post-order, depth-first search.

Prototype:

```
unsigned long
WidgetMessageSendPostOrder(tWidget *pWidget,
                           unsigned long ulMessage,
                           unsigned long ulParam1,
                           unsigned long ulParam2,
                           unsigned long bStopOnSuccess)
```

Parameters:

pWidget is a pointer to the widget tree; if this is zero then the root of the widget tree willb e used.

ulMessage is the message to send.

ulParam1 is the first parameter to the message.

ulParam2 is the second parameter to the message.

bStopOnSuccess is true if the search should be stopped when the first widget is found that returns success in response to the message.

Description:

This function performs a post-order, depth-first search of the widget tree, sending a message to each widget encountered. In a depth-first search, the children of a widget are searched before its sibling (preferring to go deeper into the tree, hence the name depth-first). A post-order search means that the message is sent to a widget after all of its children are searched.

An example use of the post-order search is for pointer-related messages; those messages should be delivered to the lowest widget in the tree before its parents (in other words, the widget deepest in the tree that has a hit should get the message, not the higher up widgets that also include the hit location).

Special handling is performed for pointer-related messages. The widget that accepts **WIDGET_MSG_PTR_DOWN** is remembered and subsequent **WIDGET_MSG_PTR_MOVE** and **WIDGET_MSG_PTR_UP** messages are sent directly to that widget.

Returns:

Returns 0 if *bStopOnSuccess* is false or no widget returned success in response to the message, or the value returned by the first widget to successfully process the message.

4.2.3.6 WidgetMessageSendPreOrder

Sends a message to a widget tree via a pre-order, depth-first search.

Prototype:

```
unsigned long
WidgetMessageSendPreOrder(tWidget *pWidget,
                          unsigned long ulMessage,
                          unsigned long ulParam1,
                          unsigned long ulParam2,
                          unsigned long bStopOnSuccess)
```

Parameters:

pWidget is a pointer to the widget tree; if this is zero then the root of the widget tree will be used.

ulMessage is the message to send.

ulParam1 is the first parameter to the message.

ulParam2 is the second parameter to the message.

bStopOnSuccess is true if the search should be stopped when the first widget is found that returns success in response to the message.

Description:

This function performs a pre-order, depth-first search of the widget tree, sending a message to each widget encountered. In a depth-first search, the children of a widget are searched before its siblings (preferring to go deeper into the tree, hence the name depth-first). A pre-order search means that the message is sent to a widget before any of its children are searched.

An example use of the pre-order search is for paint messages; the larger enclosing widgets should be drawn on the screen before the smaller widgets that reside within the parent widget (otherwise, the children would be overwritten by the parent).

Returns:

Returns 0 if *bStopOnSuccess* is false or no widget returned success in response to the message, or the value returned by the first widget to successfully process the message.

4.2.3.7 WidgetPointerMessage

Sends a pointer message.

Prototype:

```
long
WidgetPointerMessage(unsigned long ulMessage,
                     long lX,
                     long lY)
```

Parameters:

ulMessage is the pointer message to be sent.

lX is the X coordinate associated with the message.

lY is the Y coordinate associated with the message.

Description:

This function sends a pointer message to the root widget. A pointer driver (such as a touch screen driver) can use this function to deliver pointer activity to the widget tree without having to have direct knowledge of the structure of the widget framework.

Returns:

Returns 1 if the message was added to the queue, and 0 if it could not be added since the queue is full.

4.2.3.8 WidgetRemove

Removes a widget from the widget tree.

Prototype:

```
void
WidgetRemove(tWidget *pWidget)
```

Parameters:

pWidget is the widget to be removed.

Description:

This function removes a widget from the widget tree. The removed widget can be a full widget ree, allowing removal of an entire heirarchy all at once (for example, removing an entire screen from the widget tree all at once).

Returns:

None.

5 Canvas Widget

5.1 Introduction

The canvas widget provides a simple drawing surface that provides no means for interaction with the user. The canvas has the ability to be filled with a color, outlined with a color, have an image drawn in the center, have text drawn in the center, and allow the application to draw into the canvas.

When a canvas widget is drawn on the screen (via a **WIDGET_MSG_PAINT** request), the following sequence of drawing operations occurs:

- The canvas is filled with its fill color if the canvas fill style is selected. The **CANVAS_STYLE_FILL** flag enables filling of the canvas.
- The canvas is outlined with its outline color if the canvas outline style is selected. The **CANVAS_STYLE_OUTLINE** flag enables outlining of the canvas.
- The canvas image is drawn in the middle of the canvas if the canvas image style is selected. The **CANVAS_STYLE_IMG** flag enables an image on the canvas.
- The canvas text is drawn in the middle of the canvas if the canvas text style is selected. The **CANVAS_STYLE_TEXT** flag enables the text on the canvas.
- The application draws on the canvas via a callback function if the canvas application drawn style is selected. The **CANVAS_STYLE_APP_DRAWN** flag enables the application draw callback.

These steps are cumulative and any combination of these styles can be selected simultaneously. So, for example, the canvas can be filled, outlined, and then have a piece of text placed in the middle.

The canvas widget will ignore all pointer messages, making it transparent from the point of view of the pointer.

5.2 Definitions

Data Structures

- tCanvasWidget

Defines

- Canvas(sName, pParent, pNext, pChild, pDisplay, lX, lY, lWidth,lHeight, ulStyle, ulFillColor, ulOutlineColor, ulTextColor, pFont, pcText, pucImage, pfnOnPaint)

- CANVAS_STYLE_APP_DRAWN
- CANVAS_STYLE_FILL
- CANVAS_STYLE_IMG
- CANVAS_STYLE_OUTLINE
- CANVAS_STYLE_TEXT
- CANVAS_STYLE_TEXT_OPAQUE
- CanvasAppDrawnOff(pWidget)
- CanvasAppDrawnOn(pWidget)
- CanvasCallbackSet(pWidget, pfnOnPnt)
- CanvasFillColorSet(pWidget, ulColor)
- CanvasFillOff(pWidget)
- CanvasFillOn(pWidget)
- CanvasFontSet(pWidget, pFnt)
- CanvasImageOff(pWidget)
- CanvasImageOn(pWidget)
- CanvasImageSet(pWidget, pImg)
- CanvasOutlineColorSet(pWidget, ulColor)
- CanvasOutlineOff(pWidget)
- CanvasOutlineOn(pWidget)
- CanvasStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, ulStyle, ulFillColor, ulOutlineColor,ulTextColor, pFont, pcText, pucImage, pfnOnPaint)
- CanvasTextColorSet(pWidget, ulColor)
- CanvasTextOff(pWidget)
- CanvasTextOn(pWidget)
- CanvasTextOpaqueOff(pWidget)
- CanvasTextOpaqueOn(pWidget)
- CanvasTextSet(pWidget, pcTxt)

Functions

- void CanvasInit (tCanvasWidget *pWidget, const tDisplay *pDisplay, long lX, long lY, long lWidth, long lHeight)
- long CanvasMsgProc (tWidget *pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)

5.2.1 Data Structure Documentation

5.2.1.1 tCanvasWidget

Definition:

```
typedef struct
{
    tWidget sBase;
    unsigned long ulStyle;
    unsigned long ulFillColor;
```

```
        unsigned long ulOutlineColor;
        unsigned long ulTextColor;
        const tFont *pFont;
        const char *pcText;
        const unsigned char *pucImage;
        void (*pfnOnPaint)(tWidget *pWidget,
                           tContext *pContext);
    }
    tCanvasWidget
```

Members:

sBase The generic widget information.

ulStyle The style for this widget. This is a set of flags defined by CANVAS_STYLE_xxx.

ulFillColor The 24-bit RGB color used to fill this canvas, if CANVAS_STYLE_FILL is selected, and to use as the background color if CANVAS_STYLE_TEXT_OPAQUE is selected.

ulOutlineColor The 24-bit RGB color used to outline this canvas, if CANVAS_STYLE_OUTLINE is selected.

ulTextColor The 24-bit RGB color used to draw text on this canvas, if CANVAS_STYLE_TEXT is selected.

pFont A pointer to the font used to render the canvas text, if CANVAS_STYLE_TEXT is selected.

pcText A pointer to the text to draw on this canvas, if CANVAS_STYLE_TEXT is selected.

pucImage A pointer to the image to be drawn onto this canvas, if CANVAS_STYLE_IMG is selected.

pfnOnPaint A pointer to the application-supplied drawing function used to draw onto this canvas, if CANVAS_STYLE_APP_DRAWN is selected.

Description:

The structure that describes a canvas widget.

5.2.2 Define Documentation

5.2.2.1 Canvas

Declares an initialized variable containing a canvas widget data structure.

Parameters:

sName is the name of the variable to be declared.

pParent is a pointer to the parent widget.

pNext is a pointer to the sibling widget.

pChild is a pointer to the first child widget.

pDisplay is a pointer to the display on which to draw the canvas.

lX is the X coordinate of the upper left corner of the canvas.

lY is the Y coordinate of the upper left corner of the canvas.

lWidth is the width of the canvas.

lHeight is the height of the canvas.

ulStyle is the style to be applied to the canvas.

ulFillColor is the color used to fill in the canvas.

ulOutlineColor is the color used to outline the canvas.
ulTextColor is the color used to draw text on the canvas.
pFont is a pointer to the font to be used to draw text on the canvas.
pcText is a pointer to the text to draw on this canvas.
pucImage is a pointer to the image to draw on this canvas.
pfnOnPaint is a pointer to the application function to draw onto this canvas.

This macro declares a variable containing an initialized canvas widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls).

ulStyle is the logical OR of the following:

- **CANVAS_STYLE_OUTLINE** to indicate that the canvas should be outlined.
- **CANVAS_STYLE_FILL** to indicate that the canvas should be filled.
- **CANVAS_STYLE_TEXT** to indicate that the canvas should have text drawn on it (using *pFont* and *pcText*).
- **CANVAS_STYLE_IMG** to indicate that the canvas should have an image drawn on it (using *pucImage*).
- **CANVAS_STYLE_APP_DRAWN** to indicate that the canvas should be drawn with the application-supplied drawing function (using *pfnOnPaint*).
- **CANVAS_STYLE_TEXT_OPAQUE** to indicate that the canvas text should be drawn opaque (in other words, drawing the background pixels).

Returns:
Nothing; this is not a function.

5.2.2.2 CANVAS_STYLE_APP_DRAWN

This flag indicates that the canvas is drawn using the application-supplied drawing function.

5.2.2.3 CANVAS_STYLE_FILL

This flag indicates that the canvas should be filled.

5.2.2.4 CANVAS_STYLE_IMG

This flag indicates that the canvas should have an image drawn on it.

5.2.2.5 CANVAS_STYLE_OUTLINE

This flag indicates that the canvas should be outlined.

5.2.2.6 CANVAS_STYLE_TEXT

This flag indicates that the canvas should have text drawn on it.

5.2.2.7 CANVAS_STYLE_TEXT_OPAQUE

This flag indicates that the canvas text should be drawn opaque (in other words, drawing the background pixels as well as the foreground pixels).

5.2.2.8 CanvasAppDrawnOff

Disables application drawing of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function disables the use of the application callback to draw on a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.9 CanvasAppDrawnOn

Enables application drawing of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function enables the use of the application callback to draw on a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.10 CanvasCallbackSet

Sets the function to call when this canvas widget is drawn.

Parameters:
pWidget is a pointer to the canvas widget to modify.
pfnOnPnt is a pointer to the function to call.

This function sets the function to be called when this canvas is drawn and **CANVAS_STYLE_APP_DRAWN** is selected.

Returns:
None.

5.2.2.11 CanvasFillColorSet

Sets the fill color of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to be modified.
ulColor is the 24-bit RGB color to use to fill the canvas.

This function changes the color used to fill the canvas on the display. The display is not updated until the next paint request.

Returns:
None.

5.2.2.12 CanvasFillOff

Disables filling of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function disables the filling of a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.13 CanvasFillOn

Enables filling of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function enables the filling of a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.14 CanvasFontSet

Sets the font for a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.
pFnt is a pointer to the font to use to draw text on the canvas.

This function changes the font used to draw text on the canvas. The display is not updated until the next paint request.

Returns:
None.

5.2.2.15 CanvasImageOff

Disables the image on a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function disables the drawing of an image on a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.16 CanvasImageOn

Enables the image on a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function enables the drawing of an image on a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.17 CanvasImageSet

Changes the image drawn on a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to be modified.
pImg is a pointer to the image to draw onto the canvas.

This function changes the image that is drawn onto the canvas. The display is not updated until the next paint request.

Returns:
None.

5.2.2.18 CanvasOutlineColorSet

Sets the outline color of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to be modified.
ulColor is the 24-bit RGB color to use to outline the canvas.

This function changes the color used to outline the canvas on the display. The display is not updated until the next paint request.

Returns:
None.

5.2.2.19 CanvasOutlineOff

Disables outlining of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function disables the outlining of a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.20 CanvasOutlineOn

Enables outlining of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function enables the outlining of a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.21 CanvasStruct

Declares an initialized canvas widget data structure.

Parameters:
pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.

pDisplay is a pointer to the display on which to draw the canvas.
lX is the X coordinate of the upper left corner of the canvas.
lY is the Y coordinate of the upper left corner of the canvas.
lWidth is the width of the canvas.
lHeight is the height of the canvas.
ulStyle is the style to be applied to the canvas.
ulFillColor is the color used to fill in the canvas.
ulOutlineColor is the color used to outline the canvas.
ulTextColor is the color used to draw text on the canvas.
pFont is a pointer to the font to be used to draw text on the canvas.
pcText is a pointer to the text to draw on this canvas.
pucImage is a pointer to the image to draw on this canvas.
pfnOnPaint is a pointer to the application function to draw onto this canvas.

This macro provides an initialized canvas widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls). This must be assigned to a variable, such as:

```
tCanvasWidget g_sCanvas = CanvasStruct(...);
```

Or, in an array of variables:

```
tCanvasWidget g_psCanvas[] =
{
    CanvasStruct(...),
    CanvasStruct(...)
};
```

ulStyle is the logical OR of the following:

- **CANVAS_STYLE_OUTLINE** to indicate that the canvas should be outlined.
- **CANVAS_STYLE_FILL** to indicate that the canvas should be filled.
- **CANVAS_STYLE_TEXT** to indicate that the canvas should have text drawn on it (using *pFont* and *pcText*).
- **CANVAS_STYLE_IMG** to indicate that the canvas should have an image drawn on it (using *pucImage*).
- **CANVAS_STYLE_APP_DRAWN** to indicate that the canvas should be drawn with the application-supplied drawing function (using *pfnOnPaint*).
- **CANVAS_STYLE_TEXT_OPAQUE** to indicate that the canvas text should be drawn opaque (in other words, drawing the background pixels).

Returns:
Nothing; this is not a function.

5.2.2.22 CanvasTextColorSet

Sets the text color of a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to be modified.
ulColor is the 24-bit RGB color to use to draw text on the canvas.

This function changes the color used to draw text on the canvas on the display. The display is not updated until the next paint request.

Returns:
None.

5.2.2.23 CanvasTextOff

Disables the text on a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function disables the drawing of text on a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.24 CanvasTextOn

Enables the text on a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function enables the drawing of text on a canvas widget. The display is not updated until the next paint request.

Returns:
None.

5.2.2.25 CanvasTextOpaqueOff

Disables opaque text on a canvas widget.

Parameters:
pWidget is a pointer to the canvas widget to modify.

This function disables the use of opaque text on this canvas. When not using opaque text, only the foreground pixels of the text are drawn on the screen, allowing the previously drawn pixels (such as the canvas image) to show through the text.

Returns:
None.

5.2.2.26 CanvasTextOpaqueOn

Enables opaque text on a canvas widget.

Parameters:

pWidget is a pointer to the canvas widget to modify.

This function enables the use of opaque text on this canvas. When using opaque text, both the foreground and background pixels of the text are drawn on the screen, blocking out the previously drawn pixels.

Returns:

None.

5.2.2.27 CanvasTextSet

Changes the text drawn on a canvas widget.

Parameters:

pWidget is a pointer to the canvas widget to be modified.

pcTxt is a pointer to the text to draw onto the canvas.

This function changes the text that is drawn onto the canvas. The display is not updated until the next paint request.

Returns:

None.

5.2.3 Function Documentation

5.2.3.1 CanvasInit

Initializes a canvas widget.

Prototype:

```
void
CanvasInit(tCanvasWidget *pWidget,
           const tDisplay *pDisplay,
           long lX,
           long lY,
           long lWidth,
           long lHeight)
```

Parameters:

pWidget is a pointer to the canvas widget to initialize.

pDisplay is a pointer to the display on which to draw the canvas.

lX is the X coordinate of the upper left corner of the canvas.

lY is the Y coordinate of the upper left corner of the canvas.

lWidth is the width of the canvas.

lHeight is the height of the canvas.

Description:
This function initializes the provided canvas widget.

Returns:
None.

5.2.3.2 CanvasMsgProc

Handles messages for a canvas widget.

Prototype:
```
long
CanvasMsgProc(tWidget *pWidget,
              unsigned long ulMsg,
              unsigned long ulParam1,
              unsigned long ulParam2)
```

Parameters:
pWidget is a pointer to the canvas widget.
ulMsg is the message.
ulParam1 is the first parameter to the message.
ulParam2 is the second parameter to the message.

Description:
This function receives messages intended for this canvas widget and processes them accordingly. The processing of the message varies based on the message in question.

Unrecognized messages are handled by calling WidgetDefaultMsgProc().

Returns:
Returns a value appropriate to the supplied message.

6 Checkbox Widget

6.1 Introduction

The checkbox widget provides a graphical element that can be selected or unselected, resulting in a binary selection (such as "on" or "off"). A checkbox widget contains two graphical elements; the checkbox itself (which is drawn as a square that is either empty or contains an "X") and the checkbox area around the checkbox that visually indicates what the checkbox controls.

When a checkbox widget is drawn on the screen (via a **WIDGET_MSG_PAINT** request), the following sequence of drawing operations occurs:

- The checkbox area is filled with its fill color if the checkbox fill style is selected. The **CB_STYLE_FILL** flag enables filling of the checkbox area.
- The checkbox area is outlined with its outline color if the checkbox outline style is selected. The **CB_STYLE_OUTLINE** flag enables outlining of the checkbox area.
- The checkbox is drawn, either empty if it is not selected or with an "X" in the middle if it is selected.
- The checkbox image is drawn next to the checkbox if the checkbox image style is selected. The **CB_STYLE_IMG** flag enables the image next to the checkbox.
- The checkbox text is drawn next to the checkbox if the checkbox text style is selected. The **CB_STYLE_TEXT** flag enables the text next to the checkbox.

These steps are cumulative and any combination of these styles can be selected simultaneously. So, for example, the checkbox can be filled, outlined, and then have a piece of text placed next to it.

When a pointer down message is received within the extents of the checkbox area, the selected state of the checkbox is toggled. If an application callback function exists, it will be called to indicate the state change.

6.2 Definitions

Data Structures

- tCheckBoxWidget

Defines

- CB_STYLE_FILL

- CB_STYLE_IMG
- CB_STYLE_OUTLINE
- CB_STYLE_SELECTED
- CB_STYLE_TEXT
- CB_STYLE_TEXT_OPAQUE
- CheckBox(sName, pParent, pNext, pChild, pDisplay, lX, lY, lWidth,lHeight, usStyle, usBoxSize, ulFillColor, ulOutlineColor,ulTextColor, pFont, pcText, pucImage, pfnOnChange)
- CheckBoxBoxSizeSet(pWidget, usSize)
- CheckBoxCallbackSet(pWidget, pfnOnChg)
- CheckBoxFillColorSet(pWidget, ulColor)
- CheckBoxFillOff(pWidget)
- CheckBoxFillOn(pWidget)
- CheckBoxFontSet(pWidget, pFnt)
- CheckBoxImageOff(pWidget)
- CheckBoxImageOn(pWidget)
- CheckBoxImageSet(pWidget, pImg)
- CheckBoxOutlineColorSet(pWidget, ulColor)
- CheckBoxOutlineOff(pWidget)
- CheckBoxOutlineOn(pWidget)
- CheckBoxStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth,lHeight, usStyle, usBoxSize, ulFillColor,ulOutlineColor, ulTextColor, pFont, pcText, pucImage, pfnOnChange)
- CheckBoxTextColorSet(pWidget, ulColor)
- CheckBoxTextOff(pWidget)
- CheckBoxTextOn(pWidget)
- CheckBoxTextOpaqueOff(pWidget)
- CheckBoxTextOpaqueOn(pWidget)
- CheckBoxTextSet(pWidget, pcTxt)

Functions

- void CheckBoxInit (tCheckBoxWidget ∗pWidget, const tDisplay ∗pDisplay, long lX, long lY, long lWidth, long lHeight)
- long CheckBoxMsgProc (tWidget ∗pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)

6.2.1 Data Structure Documentation

6.2.1.1 tCheckBoxWidget

Definition:

```
typedef struct
{
    tWidget sBase;
    unsigned short usStyle;
    unsigned short usBoxSize;
    unsigned long ulFillColor;
```

```
        unsigned long ulOutlineColor;
        unsigned long ulTextColor;
        const tFont *pFont;
        const char *pcText;
        const unsigned char *pucImage;
        void (*pfnOnChange)(tWidget *pWidget,
                            unsigned long bSelected);
    }
    tCheckBoxWidget
```

Members:

sBase The generic widget information.

usStyle The style for this check box. This is a set of flags defined by CB_STYLE_xxx.

usBoxSize The size of the check box itself, not including the text and/or image that accompanies it (in other words, the size of the actual box that is checked or unchecked).

ulFillColor The 24-bit RGB color used to fill this check box, if CB_STYLE_FILL is selected, and to use as the background color if CB_STYLE_TEXT_OPAQUE is selected.

ulOutlineColor The 24-bit RGB color used to outline this check box, if CB_STYLE_OUTLINE is selected.

ulTextColor The 24-bit RGB color used to draw text on this check box, if CB_STYLE_TEXT is selected.

pFont The font used to draw the check box text, if CB_STYLE_TEXT is selected.

pcText A pointer to the text to draw on this check box, if CB_STYLE_TEXT is selected.

pucImage A pointer to the image to be drawn onto this check box, if CB_STYLE_IMG is selected.

pfnOnChange A pointer to the function to be called when the check box is pressed. This function is called when the state of the check box is changed.

Description:
The structure that describes a check box widget.

6.2.2 Define Documentation

6.2.2.1 CB_STYLE_FILL

Definition:
```
#define CB_STYLE_FILL
```

Description:
This flag indicates that the check box should be filled.

6.2.2.2 CB_STYLE_IMG

Definition:
```
#define CB_STYLE_IMG
```

Description:
This flag indicates that the check box should have an image drawn on it.

6.2.2.3 CB_STYLE_OUTLINE

Definition:
```
#define CB_STYLE_OUTLINE
```

Description:
This flag indicates that the check box should be outlined.

6.2.2.4 CB_STYLE_SELECTED

Definition:
```
#define CB_STYLE_SELECTED
```

Description:
This flag indicates that the check box is selected.

6.2.2.5 CB_STYLE_TEXT

Definition:
```
#define CB_STYLE_TEXT
```

Description:
This flag indicates that the check box should have text drawn on it.

6.2.2.6 CB_STYLE_TEXT_OPAQUE

Definition:
```
#define CB_STYLE_TEXT_OPAQUE
```

Description:
This flag indicates that the check box text should be drawn opaque (in other words, drawing the background pixels as well as the foreground pixels).

6.2.2.7 CheckBox

Declares an initialized variable containing a check box widget data structure.

Definition:
```
#define CheckBox(sName,
                 pParent,
                 pNext,
                 pChild,
                 pDisplay,
                 lX,
                 lY,
                 lWidth,
                 lHeight,
                 usStyle,
                 usBoxSize,
                 ulFillColor,
                 ulOutlineColor,
                 ulTextColor,
                 pFont,
                 pcText,
                 pucImage,
                 pfnOnChange)
```

Parameters:

sName is the name of the variable to be declared.
pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the check box.
lX is the X coordinate of the upper left corner of the check box.
lY is the Y coordinate of the upper left corner of the check box.
lWidth is the width of the check box.
lHeight is the height of the check box.
usStyle is the style to be applied to this check box.
usBoxSize is the size of the box that is checked.
ulFillColor is the color used to fill in the check box.
ulOutlineColor is the color used to outline the check box.
ulTextColor is the color used to draw text on the check box.
pFont is a pointer to the font to be used to draw text on the check box.
pcText is a pointer to the text to draw on this check box.
pucImage is a pointer to the image to draw on this check box.
pfnOnChange is a pointer to the function that is called when the check box is pressed.

Description:

This macro provides an initialized check box widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls).

usStyle is the logical OR of the following:

- **CB_STYLE_OUTLINE** to indicate that the check box should be outlined.
- **CB_STYLE_FILL** to indicate that the check box should be filled.
- **CB_STYLE_TEXT** to indicate that the check box should have text drawn on it (using *pFont* and *pcText*).
- **CB_STYLE_IMG** to indicate that the check box should have an image drawn on it (using *pucImage*).
- **CB_STYLE_TEXT_OPAQUE** to indicate that the check box text should be drawn opaque (in other words, drawing the background pixels).
- **CB_STYLE_SELECTED** to indicate that the check box is selected.

Returns:

Nothing; this is not a function.

6.2.2.8 CheckBoxBoxSizeSet

Sets size of the box to be checked.

Definition:

```
#define CheckBoxBoxSizeSet(pWidget,
                           usSize)
```

Parameters:

pWidget is a pointer to the check box widget to modify.

usSize is the size of the box, in pixels.

Description:

This function sets the size of the box that is drawn as part of the check box.

Returns:

None.

6.2.2.9 CheckBoxCallbackSet

Sets the function to call when this check box widget is toggled.

Definition:

```
#define CheckBoxCallbackSet(pWidget,
                            pfnOnChg)
```

Parameters:

pWidget is a pointer to the check box widget to modify.

pfnOnChg is a pointer to the function to call.

Description:

This function sets the function to be called when this check box is toggled.

Returns:

None.

6.2.2.10 CheckBoxFillColorSet

Sets the fill color of a check box widget.

Definition:

```
#define CheckBoxFillColorSet(pWidget,
                             ulColor)
```

Parameters:

pWidget is a pointer to the check box widget to be modified.

ulColor is the 24-bit RGB color to use to fill the check box.

Description:

This function changes the color used to fill the check box on the display. The display is not updated until the next paint request.

Returns:
None.

6.2.2.11 CheckBoxFillOff

Disables filling of a check box widget.

Definition:
```
#define CheckBoxFillOff(pWidget)
```

Parameters:
pWidget is a pointer to the check box widget to modify.

Description:
This function disables the filling of a check box widget. The display is not updated until the next paint request.

Returns:
None.

6.2.2.12 CheckBoxFillOn

Enables filling of a check box widget.

Definition:
```
#define CheckBoxFillOn(pWidget)
```

Parameters:
pWidget is a pointer to the check box widget to modify.

Description:
This function enables the filling of a check box widget. The display is not updated until the next paint request.

Returns:
None.

6.2.2.13 CheckBoxFontSet

Sets the font for a check box widget.

Definition:
```
#define CheckBoxFontSet(pWidget,
                        pFnt)
```

Parameters:
pWidget is a pointer to the check box widget to modify.
pFnt is a pointer to the font to use to draw text on the check box.

Description:
This function changes the font used to draw text on the check box. The display is not updated until the next paint request.

Returns:
None.

6.2.2.14 CheckBoxImageOff

Disables the image on a check box widget.

Definition:
```
#define CheckBoxImageOff(pWidget)
```

Parameters:
pWidget is a pointer to the check box widget to modify.

Description:
This function disables the drawing of an image on a check box widget. The display is not updated until the next paint request.

Returns:
None.

6.2.2.15 CheckBoxImageOn

Enables the image on a check box widget.

Definition:
```
#define CheckBoxImageOn(pWidget)
```

Parameters:
pWidget is a pointer to the check box widget to modify.

Description:
This function enables the drawing of an image on a check box widget. The display is not updated until the next paint request.

Returns:
None.

6.2.2.16 CheckBoxImageSet

Changes the image drawn on a check box widget.

Definition:
```
#define CheckBoxImageSet(pWidget,
                         pImg)
```

Parameters:

pWidget is a pointer to the check box widget to be modified.

pImg is a pointer to the image to draw onto the check box.

Description:

This function changes the image that is drawn onto the check box. The display is not updated until the next paint request.

Returns:

None.

6.2.2.17 CheckBoxOutlineColorSet

Sets the outline color of a check box widget.

Definition:

```
#define CheckBoxOutlineColorSet(pWidget,
                                ulColor)
```

Parameters:

pWidget is a pointer to the check box widget to be modified.

ulColor is the 24-bit RGB color to use to outline the check box.

Description:

This function changes the color used to outline the check box on the display. The display is not updated until the next paint request.

Returns:

None.

6.2.2.18 CheckBoxOutlineOff

Disables outlining of a check box widget.

Definition:

```
#define CheckBoxOutlineOff(pWidget)
```

Parameters:

pWidget is a pointer to the check box widget to modify.

Description:

This function disables the outlining of a check box widget. The display is not updated until the next paint request.

Returns:

None.

6.2.2.19 CheckBoxOutlineOn

Enables outlining of a check box widget.

Definition:

```
#define CheckBoxOutlineOn(pWidget)
```

Parameters:

pWidget is a pointer to the check box widget to modify.

Description:

This function enables the outlining of a check box widget. The display is not updated until the next paint request.

Returns:

None.

6.2.2.20 CheckBoxStruct

Declares an initialized check box widget data structure.

Definition:

```
#define CheckBoxStruct(pParent,
                       pNext,
                       pChild,
                       pDisplay,
                       lX,
                       lY,
                       lWidth,
                       lHeight,
                       usStyle,
                       usBoxSize,
                       ulFillColor,
                       ulOutlineColor,
                       ulTextColor,
                       pFont,
                       pcText,
                       pucImage,
                       pfnOnChange)
```

Parameters:

pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the check box.
lX is the X coordinate of the upper left corner of the check box.
lY is the Y coordinate of the upper left corner of the check box.
lWidth is the width of the check box.
lHeight is the height of the check box.
usStyle is the style to be applied to this check box.

usBoxSize is the size of the box that is checked.
ulFillColor is the color used to fill in the check box.
ulOutlineColor is the color used to outline the check box.
ulTextColor is the color used to draw text on the check box.
pFont is a pointer to the font to be used to draw text on the check box.
pcText is a pointer to the text to draw on this check box.
pucImage is a pointer to the image to draw on this check box.
pfnOnChange is a pointer to the function that is called when the check box is pressed.

Description:
This macro provides an initialized check box widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls). This must be assigned to a variable, such as:

```
tCheckBoxWidget g_sCheckBox = CheckBoxStruct(...);
```

Or, in an array of variables:

```
tCheckBoxWidget g_psCheckBoxes[] =
{
    CheckBoxStruct(...),
    CheckBoxStruct(...)
};
```

usStyle is the logical OR of the following:

- **CB_STYLE_OUTLINE** to indicate that the check box should be outlined.
- **CB_STYLE_FILL** to indicate that the check box should be filled.
- **CB_STYLE_TEXT** to indicate that the check box should have text drawn on it (using *pFont* and *pcText*).
- **CB_STYLE_IMG** to indicate that the check box should have an image drawn on it (using *pucImage*).
- **CB_STYLE_TEXT_OPAQUE** to indicate that the check box text should be drawn opaque (in other words, drawing the background pixels).
- **CB_STYLE_SELECTED** to indicate that the check box is selected.

Returns:
Nothing; this is not a function.

6.2.2.21 CheckBoxTextColorSet

Sets the text color of a check box widget.

Definition:
```
#define CheckBoxTextColorSet(pWidget,
                             ulColor)
```

Parameters:
pWidget is a pointer to the check box widget to be modified.
ulColor is the 24-bit RGB color to use to draw text on the check box.

Description:
This function changes the color used to draw text on the check box on the display. The display is not updated until the next paint request.

Returns:
None.

6.2.2.22 CheckBoxTextOff

Disables the text on a check box widget.

Definition:
```
#define CheckBoxTextOff(pWidget)
```

Parameters:
pWidget is a pointer to the check box widget to modify.

Description:
This function disables the drawing of text on a check box widget. The display is not updated until the next paint request.

Returns:
None.

6.2.2.23 CheckBoxTextOn

Enables the text on a check box widget.

Definition:
```
#define CheckBoxTextOn(pWidget)
```

Parameters:
pWidget is a pointer to the check box widget to modify.

Description:
This function enables the drawing of text on a check box widget. The display is not updated until the next paint request.

Returns:
None.

6.2.2.24 CheckBoxTextOpaqueOff

Disables opaque text on a check box widget.

Definition:
```
#define CheckBoxTextOpaqueOff(pWidget)
```

Parameters:
pWidget is a pointer to the check box widget to modify.

Description:

This function disables the use of opaque text on this check box. When not using opaque text, only the foreground pixels of the text are drawn on the screen, allowing the previously drawn pixels (such as the check box image) to show through the text.

Returns:

None.

6.2.2.25 CheckBoxTextOpaqueOn

Enables opaque text on a check box widget.

Definition:

```
#define CheckBoxTextOpaqueOn(pWidget)
```

Parameters:

pWidget is a pointer to the check box widget to modify.

Description:

This function enables the use of opaque text on this check box. When using opaque text, both the foreground and background pixels of the text are drawn on the screen, blocking out the previously drawn pixels.

Returns:

None.

6.2.2.26 CheckBoxTextSet

Changes the text drawn on a check box widget.

Definition:

```
#define CheckBoxTextSet(pWidget,
                        pcTxt)
```

Parameters:

pWidget is a pointer to the check box widget to be modified.

pcTxt is a pointer to the text to draw onto the check box.

Description:

This function changes the text that is drawn onto the check box. The display is not updated until the next paint request.

Returns:

None.

6.2.3 Function Documentation

6.2.3.1 CheckBoxInit

Initializes a check box widget.

Prototype:

```
void
CheckBoxInit(tCheckBoxWidget *pWidget,
             const tDisplay *pDisplay,
             long lX,
             long lY,
             long lWidth,
             long lHeight)
```

Parameters:

pWidget is a pointer to the check box widget to initialize.

pDisplay is a pointer to the display on which to draw the check box.

lX is the X coordinate of the upper left corner of the check box.

lY is the Y coordinate of the upper left corner of the check box.

lWidth is the width of the check box.

lHeight is the height of the check box.

Description:

This function initializes the provided check box widget.

Returns:

None.

6.2.3.2 CheckBoxMsgProc

Handles messages for a check box widget.

Prototype:

```
long
CheckBoxMsgProc(tWidget *pWidget,
                unsigned long ulMsg,
                unsigned long ulParam1,
                unsigned long ulParam2)
```

Parameters:

pWidget is a pointer to the check box widget.

ulMsg is the message.

ulParam1 is the first parameter to the message.

ulParam2 is the second parameter to the message.

Description:

This function receives messages intended for this check box widget and processes them accordingly. The processing of the message varies based on the message in question.

Unrecognized messages are handled by calling WidgetDefaultMsgProc().

Returns:

Returns a value appropriate to the supplied message.

7 Container Widget

7.1 Introduction

The container widget provides means of grouping widget together within the widget heirarchy, most notably useful for joining together several radio button widgets to provide a single one-of selection. The container widget can also provide a visual grouping of the child widgets by drawing a box around the widget area.

When a container widget is drawn on the screen (via a **WIDGET_MSG_PAINT** request), the following sequence of drawing operations occurs:

- The container is filled with its fill color if the container fill style is selected. The **CTR_STYLE_FILL** flag enables filling of the container.
- The container text is drawn at the top of the container if the container text style is selected. The **CTR_STYLE_TEXT** flag enables the text on the container. The text is drawn centered horizontally if the **CTR_STYLE_TEXT_CENTER** flag is selected; otherwise the text is drawn on the left side of the widget.
- The container is outlined with its outline color if the container outline style is selected. The **CTR_STYLE_OUTLINE** flag enables outlining of the container.

These steps are cumulative and any combination of these styles can be selected simultaneously.

The container widget will ignore all pointer messages, making it transparent from the point of view of the pointer.

7.2 Definitions

Data Structures

- tContainerWidget

Defines

- Container(sName, pParent, pNext, pChild, pDisplay, lX, lY, lWidth,lHeight, ulStyle, ulFillColor, ulOutlineColor, ulTextColor, pFont, pcText)
- ContainerFillColorSet(pWidget, ulColor)
- ContainerFillOff(pWidget)
- ContainerFillOn(pWidget)
- ContainerFontSet(pWidget, pFnt)
- ContainerOutlineColorSet(pWidget, ulColor)

- ContainerOutlineOff(pWidget)
- ContainerOutlineOn(pWidget)
- ContainerStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, ulStyle, ulFillColor, ulOutlineColor,ulTextColor, pFont, pcText)
- ContainerTextCenterOff(pWidget)
- ContainerTextCenterOn(pWidget)
- ContainerTextColorSet(pWidget, ulColor)
- ContainerTextOff(pWidget)
- ContainerTextOn(pWidget)
- ContainerTextOpaqueOff(pWidget)
- ContainerTextOpaqueOn(pWidget)
- ContainerTextSet(pWidget, pcTxt)
- CTR_STYLE_FILL
- CTR_STYLE_OUTLINE
- CTR_STYLE_TEXT
- CTR_STYLE_TEXT_CENTER
- CTR_STYLE_TEXT_OPAQUE

Functions

- void ContainerInit (tContainerWidget ∗pWidget, const tDisplay ∗pDisplay, long lX, long lY, long lWidth, long lHeight)
- long ContainerMsgProc (tWidget ∗pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)

7.2.1 Data Structure Documentation

7.2.1.1 tContainerWidget

Definition:

```
typedef struct
{
    tWidget sBase;
    unsigned long ulStyle;
    unsigned long ulFillColor;
    unsigned long ulOutlineColor;
    unsigned long ulTextColor;
    const tFont *pFont;
    const char *pcText;
}
tContainerWidget
```

Members:

sBase The generic widget information.

ulStyle The style for this widget. This is a set of flags defined by CTR_STYLE_xxx.

ulFillColor The 24-bit RGB color used to fill this container widget, if CTR_STYLE_FILL is selected, and to use as the background color if CTR_STYLE_TEXT_OPAQUE is selected.

ulOutlineColor The 24-bit RGB color used to outline this container widget, if CTR_STYLE_OUTLINE is selected.

ulTextColor The 24-bit RGB color used to draw text on this container widget, if CTR_STYLE_TEXT is selected.

pFont A pointer to the font used to render the container text, if CTR_STYLE_TEXT is selected.

pcText A pointer to the text to draw on this container widget, if CTR_STYLE_TEXT is selected.

Description:

The structure that describes a container widget.

7.2.2 Define Documentation

7.2.2.1 Container

Declares an initialized variable containing a container widget data structure.

Definition:

```
#define Container(sName,
                  pParent,
                  pNext,
                  pChild,
                  pDisplay,
                  lX,
                  lY,
                  lWidth,
                  lHeight,
                  ulStyle,
                  ulFillColor,
                  ulOutlineColor,
                  ulTextColor,
                  pFont,
                  pcText)
```

Parameters:

sName is the name of the variable to be declared.

pParent is a pointer to the parent widget.

pNext is a pointer to the sibling widget.

pChild is a pointer to the first child widget.

pDisplay is a pointer to the display on which to draw the container widget.

lX is the X coordinate of the upper left corner of the container widget.

lY is the Y coordinate of the upper left corner of the container widget.

lWidth is the width of the container widget.

lHeight is the height of the container widget.

ulStyle is the style to be applied to the container widget.

ulFillColor is the color used to fill in the container widget.

ulOutlineColor is the color used to outline the container widget.

ulTextColor is the color used to draw text on the container widget.

pFont is a pointer to the font to be used to draw text on the container widget.

pcText is a poitner to the text to draw on the container widget.

Description:

This macro provides an initialized container widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls).

ulStyle is the logical OR of the following:

- **CTR_STYLE_OUTLINE** to indicate that the container widget should be outlined.
- **CTR_STYLE_FILL** to indicate that the container widget should be filled.
- **CTR_STYLE_TEXT** to indicate that the container widget should have text drawn on it (using *pFont* and *pcText*).
- **CTR_STYLE_TEXT_OPAQUE** to indicate that the container widget text should be drawn opaque (in other words, drawing the background pixels).
- **CTR_STYLE_TEXT_CENTER** to indicate that the container widget text should be drawn centered horizontally.

Returns:

Nothing; this is not a function.

7.2.2.2 ContainerFillColorSet

Sets the fill color of a container widget.

Definition:

```
#define ContainerFillColorSet(pWidget,
                                  ulColor)
```

Parameters:

pWidget is a pointer to the container widget to be modified.

ulColor is the 24-bit RGB color to use to fill the container widget.

Description:

This function changes the color used to fill the container widget on the display. The display is not updated until the next paint request.

Returns:

None.

7.2.2.3 ContainerFillOff

Disables filling of a container widget.

Definition:

```
#define ContainerFillOff(pWidget)
```

Parameters:

pWidget is a pointer to the container widget to modify.

Description:

This function disables the filling of a container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.4 ContainerFillOn

Enables filling of a container widget.

Definition:
```
#define ContainerFillOn(pWidget)
```

Parameters:
pWidget is a pointer to the container widget to modify.

Description:
This function enables the filling of a container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.5 ContainerFontSet

Sets the font for a container widget.

Definition:
```
#define ContainerFontSet(pWidget,
                         pFnt)
```

Parameters:
pWidget is a pointer to the container widget to modify.
pFnt is a pointer to the font to use to draw text on the container widget.

Description:
This function changes the font used to draw text on the container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.6 ContainerOutlineColorSet

Sets the outline color of a container widget.

Definition:
```
#define ContainerOutlineColorSet(pWidget,
                                 ulColor)
```

Parameters:
pWidget is a pointer to the container widget to be modified.

ulColor is the 24-bit RGB color to use to outline the container widget.

Description:
This function changes the color used to outline the container widget on the display. The display is not updated until the next paint request.

Returns:
None.

7.2.2.7 ContainerOutlineOff

Disables outlining of a container widget.

Definition:
```
#define ContainerOutlineOff(pWidget)
```

Parameters:
pWidget is a pointer to the container widget to modify.

Description:
This function disables the outlining of a container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.8 ContainerOutlineOn

Enables outlining of a container widget.

Definition:
```
#define ContainerOutlineOn(pWidget)
```

Parameters:
pWidget is a pointer to the container widget to modify.

Description:
This function enables the outlining of a container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.9 ContainerStruct

Declares an initialized container widget data structure.

Definition:

```
#define ContainerStruct(pParent,
                        pNext,
                        pChild,
                        pDisplay,
                        lX,
                        lY,
                        lWidth,
                        lHeight,
                        ulStyle,
                        ulFillColor,
                        ulOutlineColor,
                        ulTextColor,
                        pFont,
                        pcText)
```

Parameters:

pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the container widget.
lX is the X coordinate of the upper left corner of the container widget.
lY is the Y coordinate of the upper left corner of the container widget.
lWidth is the width of the container widget.
lHeight is the height of the container widget.
ulStyle is the style to be applied to the container widget.
ulFillColor is the color used to fill in the container widget.
ulOutlineColor is the color used to outline the container widget.
ulTextColor is the color used to draw text on the container widget.
pFont is a pointer to the font to be used to draw text on the container widget.
pcText is a poitner to the text to draw on the container widget.

Description:

This macro provides an initialized container widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls). This must be assigned to a variable, such as:

```
tContainerWidget g_sContainer = ContainerStruct(...);
```

Or, in an array of variables:

```
tContainerWidget g_psContainers[] =
{
    ContainerStruct(...),
    ContainerStruct(...)
};
```

ulStyle is the logical OR of the following:

- **CTR_STYLE_OUTLINE** to indicate that the container widget should be outlined.
- **CTR_STYLE_FILL** to indicate that the container widget should be filled.
- **CTR_STYLE_TEXT** to indicate that the container widget should have text drawn on it (using *pFont* and *pcText*).

- **CTR_STYLE_TEXT_OPAQUE** to indicate that the container widget text should be drawn opaque (in other words, drawing the background pixels).
- **CTR_STYLE_TEXT_CENTER** to indicate that the container widget text should be drawn centered horizontally.

Returns:
Nothing; this is not a function.

7.2.2.10 ContainerTextCenterOff

Disables the centering of text on a container widget.

Definition:
```
#define ContainerTextCenterOff(pWidget)
```

Parameters:
pWidget is a pointer to the container widget to modify.

Description:
This function disables the centering of text on a container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.11 ContainerTextCenterOn

Enables the centering of text on a container widget.

Definition:
```
#define ContainerTextCenterOn(pWidget)
```

Parameters:
pWidget is a pointer to the container widget to modify.

Description:
This function enables the centering of text on a container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.12 ContainerTextColorSet

Sets the text color of a container widget.

Definition:
```
#define ContainerTextColorSet(pWidget,
                              ulColor)
```

Parameters:

pWidget is a pointer to the container widget to be modified.

ulColor is the 24-bit RGB color to use to draw text on the container widget.

Description:

This function changes the color used to draw text on the container widget on the display. The display is not updated until the next paint request.

Returns:

None.

7.2.2.13 ContainerTextOff

Disables the text on a container widget.

Definition:

```
#define ContainerTextOff(pWidget)
```

Parameters:

pWidget is a pointer to the container widget to modify.

Description:

This function disables the drawing of text on a container widget. The display is not updated until the next paint request.

Returns:

None.

7.2.2.14 ContainerTextOn

Enables the text on a container widget.

Definition:

```
#define ContainerTextOn(pWidget)
```

Parameters:

pWidget is a pointer to the container widget to modify.

Description:

This function enables the drawing of text on a container widget. The display is not updated until the next paint request.

Returns:

None.

7.2.2.15 ContainerTextOpaqueOff

Disables opaque text on a container widget.

Definition:
```
#define ContainerTextOpaqueOff(pWidget)
```

Parameters:
pWidget is a pointer to the container widget to modify.

Description:
This function disables the use of opaque text on this container widget. When not using opaque text, only the foreground pixels of the text are drawn on the screen, allowing the previously drawn pixels (such as the background) to show through the text.

Returns:
None.

7.2.2.16 ContainerTextOpaqueOn

Enables opaque text on a container widget.

Definition:
```
#define ContainerTextOpaqueOn(pWidget)
```

Parameters:
pWidget is a pointer to the container widget to modify.

Description:
This function enables the use of opaque text on this container widget. When using opaque text, both the foreground and background pixels of the text are drawn on the screen, blocking out the previously drawn pixels.

Returns:
None.

7.2.2.17 ContainerTextSet

Changes the text drawn on a container widget.

Definition:
```
#define ContainerTextSet(pWidget,
                         pcTxt)
```

Parameters:
pWidget is a pointer to the container widget to be modified.
pcTxt is a pointer to the text to draw onto the container widget.

Description:
This function changes the text that is drawn onto the container widget. The display is not updated until the next paint request.

Returns:
None.

7.2.2.18 CTR_STYLE_FILL

Definition:

```
#define CTR_STYLE_FILL
```

Description:

This flag indicates that the container widget should be filled.

7.2.2.19 CTR_STYLE_OUTLINE

Definition:

```
#define CTR_STYLE_OUTLINE
```

Description:

This flag indicates that the container widget should be outlined.

7.2.2.20 CTR_STYLE_TEXT

Definition:

```
#define CTR_STYLE_TEXT
```

Description:

This flag indicates that the container widget should have text drawn on it.

7.2.2.21 CTR_STYLE_TEXT_CENTER

Definition:

```
#define CTR_STYLE_TEXT_CENTER
```

Description:

This flag indicates that the container text should be drawn centered within the width of the container.

7.2.2.22 CTR_STYLE_TEXT_OPAQUE

Definition:

```
#define CTR_STYLE_TEXT_OPAQUE
```

Description:

This flag indicates that the container text should be drawn opaque (in other words, drawing the background pixels as well as the foreground pixels).

7.2.3 Function Documentation

7.2.3.1 ContainerInit

Initializes a container widget.

Prototype:

```
void
ContainerInit(tContainerWidget *pWidget,
              const tDisplay *pDisplay,
              long lX,
              long lY,
              long lWidth,
              long lHeight)
```

Parameters:

pWidget is a pointer to the container widget to initialize.

pDisplay is a pointer to the display on which to draw the container widget.

lX is the X coordinate of the upper left corner of the container widget.

lY is the Y coordinate of the upper left corner of the container widget.

lWidth is the width of the container widget.

lHeight is the height of the container widget.

Description:

This function initializes a container widget, preparing it for placement into the widget tree.

Returns:

none.

7.2.3.2 ContainerMsgProc

Handles messages for a container widget.

Prototype:

```
long
ContainerMsgProc(tWidget *pWidget,
                 unsigned long ulMsg,
                 unsigned long ulParam1,
                 unsigned long ulParam2)
```

Parameters:

pWidget is a pointer to the container widget.

ulMsg is the message.

ulParam1 is the first parameter to the message.

ulParam2 is the second parameter to the message.

Description:

This function receives messages intended for this container widget and processes them accordingly. The processing of the message varies based on the message in question.

Unrecognized messages are handled by calling WidgetDefaultMsgProc().

Returns:

Returns a value appropriate to the supplied message.

8 Push Button Widget

8.1 Introduction

The push button widget provides a button that can be pressed, causing an action to be performed. A push button has the ability to be filled iwth a color, outlined with a color, have an image drawn in the center, and have text drawn in the center. Two fill colors and two images can be utilized to provide a visual indication of the pressed or released state of the push button.

Push button widgets can be rectangular or circular. Circular push buttons are not necessarily circular when drawn; the image or text may extend beyond the extents of the circle. But, circular push buttons will only accept pointer events that reside within the extent of the cirle. Rectangular push buttons accept pointer events that reside within the extent of the enclosing rectangle.

When a push button widget is drawn on the screen (via a **WIDGET_MSG_PAINT** request), the following sequence of drawing operations occurs:

- The push button is filled with its fill color if the push button fill style is selected. The **PB_STYLE_FILL** flag enables filling of the push button.
- The push button is outlined with its outline color if the push button outline style is selected. The **PB_STYLE_OUTLINE** flag enables outlining of the push button.
- The push button image is drawn in the middle of the push button if the push button image style is selected. The **PB_STYLE_IMG** flag enables an image on the push button.
- The push button text is drawn in the middle of the push button if the push button text style is selected. The **PB_STYLE_TEXT** flag enables the text on the push button.

These steps are cumulative and any combination of these styles can be selected simultaneously. So, for example, the push button can be filled, outlined, and then have a piece of text placed in the middle.

When a pointer down message is received within the extents of the push button, the application callback function is called if present. An auto-repeat capability can be enabled, which will call the application callback at a periodic rate after an initial press delay so long as the pointer remains within the extents of the push button.

In addition to the application callback, the visual appearance of the push button is also changed when a pointer down or pointer up message is received (depending on the style of the push button).

8.2 Definitions

Data Structures

- tPushButtonWidget

Defines

- CircularButton(sName, pParent, pNext, pChild, pDisplay, lX, lY, lR,ulStyle, ulFillColor, ulPressFillColor,ulOutlineColor, ulTextColor, pFont, pcText, pucImage, pucPressImage, usAutoRepeatDelay, usAutoRepeatRate,pfnOnClick)
- CircularButtonStruct(pParent, pNext, pChild, pDisplay, lX, lY, lR, ulStyle, ulFillColor, ulPressFillColor,ulOutlineColor, ulTextColor, pFont, pcText,pucImage, pucPressImage, usAutoRepeatDelay,usAutoRepeatRate, pfnOnClick)
- PB_STYLE_AUTO_REPEAT
- PB_STYLE_FILL
- PB_STYLE_IMG
- PB_STYLE_OUTLINE
- PB_STYLE_PRESSED
- PB_STYLE_TEXT
- PB_STYLE_TEXT_OPAQUE
- PushButtonAutoRepeatDelaySet(pWidget, usDelay)
- PushButtonAutoRepeatOff(pWidget)
- PushButtonAutoRepeatOn(pWidget)
- PushButtonAutoRepeatRateSet(pWidget, usRate)
- PushButtonCallbackSet(pWidget, pfnOnClik)
- PushButtonFillColorPressedSet(pWidget, ulColor)
- PushButtonFillColorSet(pWidget, ulColor)
- PushButtonFillOff(pWidget)
- PushButtonFillOn(pWidget)
- PushButtonFontSet(pWidget, pFnt)
- PushButtonImageOff(pWidget)
- PushButtonImageOn(pWidget)
- PushButtonImagePressedSet(pWidget, pImg)
- PushButtonImageSet(pWidget, pImg)
- PushButtonOutlineColorSet(pWidget, ulColor)
- PushButtonOutlineOff(pWidget)
- PushButtonOutlineOn(pWidget)
- PushButtonTextColorSet(pWidget, ulColor)
- PushButtonTextOff(pWidget)
- PushButtonTextOn(pWidget)
- PushButtonTextOpaqueOff(pWidget)
- PushButtonTextOpaqueOn(pWidget)
- PushButtonTextSet(pWidget, pcTxt)
- RectangularButton(sName, pParent, pNext, pChild, pDisplay, lX, lY,lWidth, lHeight, ulStyle, ulFillColor,ulPressFillColor, ulOutlineColor, ulTextColor,pFont, pcText, pucImage, pucPressImage,usAutoRepeatDelay, usAutoRepeatRate, pfnOnClick)
- RectangularButtonStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, ulStyle, ulFillColor,ulPressFillColor, ulOutlineColor,ulTextColor, pFont, pcText, pucImage,pucPressImage, usAutoRepeatDelay,usAutoRepeatRate, pfnOnClick)

Functions

- void CircularButtonInit (tPushButtonWidget ∗pWidget, const tDisplay ∗pDisplay, long lX, long lY, long lR)
- long CircularButtonMsgProc (tWidget ∗pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)
- void RectangularButtonInit (tPushButtonWidget ∗pWidget, const tDisplay ∗pDisplay, long lX, long lY, long lWidth, long lHeight)
- long RectangularButtonMsgProc (tWidget ∗pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)

8.2.1 Data Structure Documentation

8.2.1.1 tPushButtonWidget

Definition:

```
typedef struct
{
    tWidget sBase;
    unsigned long ulStyle;
    unsigned long ulFillColor;
    unsigned long ulPressFillColor;
    unsigned long ulOutlineColor;
    unsigned long ulTextColor;
    const tFont *pFont;
    const char *pcText;
    const unsigned char *pucImage;
    const unsigned char *pucPressImage;
    unsigned short usAutoRepeatDelay;
    unsigned short usAutoRepeatRate;
    unsigned long ulAutoRepeatCount;
    void (*pfnOnClick)(tWidget *pWidget);
}
tPushButtonWidget
```

Members:

sBase The generic widget information.

ulStyle The style for this widget. This is a set of flags defined by PB_STYLE_xxx.

ulFillColor The 24-bit RGB color used to fill this push button, if PB_STYLE_FILL is selected, and to use as the background color if PB_STYLE_TEXT_OPAQUE is selected.

ulPressFillColor The 24-bit RGB color used to fill this push button when it is pressed, if PB_STYLE_FILL is selected, and to use as the background color if PB_STYLE_TEXT_OPAQUE is selected.

ulOutlineColor The 24-bit RGB color used to outline this push button, if PB_STYLE_OUTLINE is selected.

ulTextColor The 24-bit RGB color used to draw text on this push button, if PB_STYLE_TEXT is selected.

pFont A pointer to the font used to render the push button text, if PB_STYLE_TEXT is selected.

pcText A pointer to the text to draw on this push button, if PB_STYLE_TEXT is selected.

pucImage A pointer to the image to be drawn onto this push button, if PB_STYLE_IMG is selected.

pucPressImage A pointer to the image to be drawn onto this push button when it is pressed, if PB_STYLE_IMG is selected.

usAutoRepeatDelay The number of pointer events to delay before starting to auto-repeat, if PB_STYLE_AUTO_REPEAT is selected. The amount of time to which this corresponds is dependent upon the rate at which pointer events are generated by the pointer driver.

usAutoRepeatRate The number of pointer events between button presses generated by the auto-repeat function, if PB_STYLE_AUTO_REPEAT is selected. The amount of time to which this corresponds is dependent up on the rate at which pointer events are generated by the pointer driver.

ulAutoRepeatCount The number of pointer events that have occurred. This is used when PB_STYLE_AUTO_REPEAT is selected to generate the auto-repeat events.

pfnOnClick A pointer to the function to be called when the button is pressed. This is repeatedly called when PB_STYLE_AUTO_REPEAT is selected.

Description:
The structure that describes a push button widget.

8.2.2 Define Documentation

8.2.2.1 CircularButton

Declares an initialized variable containing a circular push button widget data structure.

Definition:
```
#define CircularButton(sName,
                       pParent,
                       pNext,
                       pChild,
                       pDisplay,
                       lX,
                       lY,
                       lR,
                       ulStyle,
                       ulFillColor,
                       ulPressFillColor,
                       ulOutlineColor,
                       ulTextColor,
                       pFont,
                       pcText,
                       pucImage,
                       pucPressImage,
                       usAutoRepeatDelay,
                       usAutoRepeatRate,
                       pfnOnClick)
```

Parameters:

sName is the name of the variable to be declared.

pParent is a pointer to the parent widget.

pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the push button.
lX is the X coordinate of the center of the push button.
lY is the Y coordinate of the center of the push button.
lR is the radius of the push button.
ulStyle is the style to be applied to the push button.
ulFillColor is the color used to fill in the push button.
ulPressFillColor is the color used to fill in the push button when it is pressed.
ulOutlineColor is the color used to outline the push button.
ulTextColor is the color used to draw text on the push button.
pFont is a pointer to the font to be used to draw text on the push button.
pcText is a pointer to the text to draw on this push button.
pucImage is a pointer to the image to draw on this push button.
pucPressImage is a pointer to the image to draw on this push button when it is pressed.
usAutoRepeatDelay is the delay before starting auto-repeat.
usAutoRepeatRate is the rate at which auto-repeat events are generated.
pfnOnClick is a pointer to the function that is called when the push button is pressed.

Description:

This macro provides an initialized circular push button widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls).

ulStyle is the logical OR of the following:

- **PB_STYLE_OUTLINE** to indicate that the push button should be outlined.
- **PB_STYLE_FILL** to indicate that the push button should be filled.
- **PB_STYLE_TEXT** to indicate that the push button should have text drawn on it (using *pFont* and *pcText*).
- **PB_STYLE_IMG** to indicate that the push button should have an image drawn on it (using *pucImage*).
- **PB_STYLE_TEXT_OPAQUE** to indicate that the push button text should be drawn opaque (in other words, drawing the background pixels).
- **PB_STYLE_AUTO_REPEAT** to indicate that auto-repeat should be used.

Returns:

Nothing; this is not a function.

8.2.2.2 CircularButtonStruct

Declares an initialized circular push button widget data structure.

Definition:

```
#define CircularButtonStruct(pParent,
                             pNext,
                             pChild,
                             pDisplay,
                             lX,
                             lY,
                             lR,
                             ulStyle,
                             ulFillColor,
                             ulPressFillColor,
                             ulOutlineColor,
                             ulTextColor,
                             pFont,
                             pcText,
                             pucImage,
                             pucPressImage,
                             usAutoRepeatDelay,
                             usAutoRepeatRate,
                             pfnOnClick)
```

Parameters:

pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the push button.
lX is the X coordinate of the center of the push button.
lY is the Y coordinate of the center of the push button.
lR is the radius of the push button.
ulStyle is the style to be applied to the push button.
ulFillColor is the color used to fill in the push button.
ulPressFillColor is the color used to fill in the push button when it is pressed.
ulOutlineColor is the color used to outline the push button.
ulTextColor is the color used to draw text on the push button.
pFont is a pointer to the font to be used to draw text on the push button.
pcText is a pointer to the text to draw on this push button.
pucImage is a pointer to the image to draw on this push button.
pucPressImage is a pointer to the image to draw on this push button when it is pressed.
usAutoRepeatDelay is the delay before starting auto-repeat.
usAutoRepeatRate is the rate at which auto-repeat events are generated.
pfnOnClick is a pointer to the function that is called when the push button is pressed.

Description:

This macro provides an initialized circular push button widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls). This must be assigned to a variable, such as:

```
tPushButtonWidget g_sPushButton = CircularButtonStruct(...);
```

Or, in an array of variables:

```
tPushButtonWidget g_psPushButtons[] =
{
    CircularButtonStruct(...),
    CircularButtonStruct(...)
};
```

ulStyle is the logical OR of the following:

- **PB_STYLE_OUTLINE** to indicate that the push button should be outlined.
- **PB_STYLE_FILL** to indicate that the push button should be filled.
- **PB_STYLE_TEXT** to indicate that the push button should have text drawn on it (using *pFont* and *pcText*).
- **PB_STYLE_IMG** to indicate that the push button should have an image drawn on it (using *pucImage*).
- **PB_STYLE_TEXT_OPAQUE** to indicate that the push button text should be drawn opaque (in other words, drawing the background pixels).
- **PB_STYLE_AUTO_REPEAT** to indicate that auto-repeat should be used.

Returns:
Nothing; this is not a function.

8.2.2.3 PB_STYLE_AUTO_REPEAT

Definition:
```
#define PB_STYLE_AUTO_REPEAT
```

Description:
This flag indicates that the push button should auto-repeat, generating repeated click events while it is pressed.

8.2.2.4 PB_STYLE_FILL

Definition:
```
#define PB_STYLE_FILL
```

Description:
This flag indicates that the push button should be filled.

8.2.2.5 PB_STYLE_IMG

Definition:
```
#define PB_STYLE_IMG
```

Description:
This flag indicates that the push button should have an image drawn on it.

8.2.2.6 PB_STYLE_OUTLINE

Definition:
```
#define PB_STYLE_OUTLINE
```

Description:
This flag indicates that the push button should be outlined.

8.2.2.7 PB_STYLE_PRESSED

Definition:

```
#define PB_STYLE_PRESSED
```

Description:
This flag indicates that the push button is pressed.

8.2.2.8 PB_STYLE_TEXT

Definition:

```
#define PB_STYLE_TEXT
```

Description:
This flag indicates that the push button should have text drawn on it.

8.2.2.9 PB_STYLE_TEXT_OPAQUE

Definition:

```
#define PB_STYLE_TEXT_OPAQUE
```

Description:
This flag indicates that the push button text should be drawn opaque (in other words, drawing the background pixels as well as the foreground pixels).

8.2.2.10 PushButtonAutoRepeatDelaySet

Sets the auto-repeat delay for a push button widget.

Definition:

```
#define PushButtonAutoRepeatDelaySet(pWidget,
                                     usDelay)
```

Parameters:
pWidget is a pointer to the push button widget to modify.
usDelay is the number of pointer events before auto-repeat starts.

Description:
This function sets the delay before auto-repeat begins. Unpredictable behavior will occur if this is called while the push button is pressed.

Returns:
None.

8.2.2.11 PushButtonAutoRepeatOff

Disables auto-repeat for a push button widget.

Definition:
```
#define PushButtonAutoRepeatOff(pWidget)
```

Parameters:
pWidget is a pointer to the push button widget to modify.

Description:
This function disables the auto-repeat behavior of a push button.

Returns:
None.

8.2.2.12 PushButtonAutoRepeatOn

Enables auto-repeat for a push button widget.

Definition:
```
#define PushButtonAutoRepeatOn(pWidget)
```

Parameters:
pWidget is a pointer to the push button widget to modify.

Description:
This function enables the auto-repeat behavior of a push button. Unpredictable behavior will occur if this is called while the push button is pressed.

Returns:
None.

8.2.2.13 PushButtonAutoRepeatRateSet

Sets the auto-repeat rate for a push button widget.

Definition:
```
#define PushButtonAutoRepeatRateSet(pWidget,
                                    usRate)
```

Parameters:
pWidget is a pointer to the push button widget to modify.
usRate is the number of pointer events between auto-repeat events.

Description:
This function sets the rate at which auto-repeat events occur. Unpredictable behavior will occur if this is called while the push button is pressed.

Returns:
None.

8.2.2.14 PushButtonCallbackSet

Sets the function to call when this push button widget is pressed.

Definition:

```
#define PushButtonCallbackSet(pWidget,
                              pfnOnClik)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

pfnOnClik is a pointer to the function to call.

Description:

This function sets the function to be called when this push button is pressed. The supplied function is called when the push button is first pressed, and then repeated while the push button is pressed if auto-repeat is enabled.

Returns:

None.

8.2.2.15 PushButtonFillColorPressedSet

Sets the fill color of a push button widget when it is pressed.

Definition:

```
#define PushButtonFillColorPressedSet(pWidget,
                                      ulColor)
```

Parameters:

pWidget is a pointer to the push button widget to be modified.

ulColor is the 24-bit RGB color to use to fill the push button when it is pressed.

Description:

This function changes the color used to fill the push button on the display when it is pressed. The display is not updated until the next paint request.

Returns:

None.

8.2.2.16 PushButtonFillColorSet

Sets the fill color of a push button widget.

Definition:

```
#define PushButtonFillColorSet(pWidget,
                               ulColor)
```

Parameters:

pWidget is a pointer to the push button widget to be modified.

ulColor is the 24-bit RGB color to use to fill the push button.

Description:
This function changes the color used to fill the push button on the display. The display is not updated until the next paint request.

Returns:
None.

8.2.2.17 PushButtonFillOff

Disables filling of a push button widget.

Definition:
```
#define PushButtonFillOff(pWidget)
```

Parameters:
pWidget is a pointer to the push button widget to modify.

Description:
This function disables the filling of a push button widget. The display is not updated until the next paint request.

Returns:
None.

8.2.2.18 PushButtonFillOn

Enables filling of a push button widget.

Definition:
```
#define PushButtonFillOn(pWidget)
```

Parameters:
pWidget is a pointer to the push button widget to modify.

Description:
This function enables the filling of a push button widget. The display is not updated until the next paint request.

Returns:
None.

8.2.2.19 PushButtonFontSet

Sets the font for a push button widget.

Definition:
```
#define PushButtonFontSet(pWidget,
                          pFnt)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

pFnt is a pointer to the font to use to draw text on the push button.

Description:

This function changes the font used to draw text on the push button. The display is not updated until the next paint request.

Returns:

None.

8.2.2.20 PushButtonImageOff

Disables the image on a push button widget.

Definition:

```
#define PushButtonImageOff(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function disables the drawing of an image on a push button widget. The display is not updated until the next paint request.

Returns:

None.

8.2.2.21 PushButtonImageOn

Enables the image on a push button widget.

Definition:

```
#define PushButtonImageOn(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function enables the drawing of an image on a push button widget. The display is not updated until the next paint request.

Returns:

None.

8.2.2.22 PushButtonImagePressedSet

Changes the image drawn on a push button widget when it is pressed.

Definition:

```
#define PushButtonImagePressedSet(pWidget,
                                  pImg)
```

Parameters:

pWidget is a pointer to the push button widget to be modified.

pImg is a pointer to the image to draw onto the push button when it is pressed.

Description:

This function changes the image that is drawn onto the push button when it is pressed. The display is not updated until the next paint request.

Returns:

None.

8.2.2.23 PushButtonImageSet

Changes the image drawn on a push button widget.

Definition:

```
#define PushButtonImageSet(pWidget,
                           pImg)
```

Parameters:

pWidget is a pointer to the push button widget to be modified.

pImg is a pointer to the image to draw onto the push button.

Description:

This function changes the image that is drawn onto the push button. The display is not updated until the next paint request.

Returns:

None.

8.2.2.24 PushButtonOutlineColorSet

Sets the outline color of a push button widget.

Definition:

```
#define PushButtonOutlineColorSet(pWidget,
                                  ulColor)
```

Parameters:

pWidget is a pointer to the push button widget to be modified.

ulColor is the 24-bit RGB color to use to outline the push button.

Description:

This function changes the color used to outline the push button on the display. The display is not updated until the next paint request.

Returns:

None.

8.2.2.25 PushButtonOutlineOff

Disables outlining of a push button widget.

Definition:

```
#define PushButtonOutlineOff(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function disables the outlining of a push button widget. The display is not updated until the next paint request.

Returns:

None.

8.2.2.26 PushButtonOutlineOn

Enables outlining of a push button widget.

Definition:

```
#define PushButtonOutlineOn(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function enables the outlining of a push button widget. The display is not updated until the next paint request.

Returns:

None.

8.2.2.27 PushButtonTextColorSet

Sets the text color of a push button widget.

Definition:

```
#define PushButtonTextColorSet(pWidget,
                               ulColor)
```

Parameters:

pWidget is a pointer to the push button widget to be modified.

ulColor is the 24-bit RGB color to use to draw text on the push button.

Description:

This function changes the color used to draw text on the push button on the display. The display is not updated until the next paint request.

Returns:

None.

8.2.2.28 PushButtonTextOff

Disables the text on a push button widget.

Definition:

```
#define PushButtonTextOff(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function disables the drawing of text on a push button widget. The display is not updated until the next paint request.

Returns:

None.

8.2.2.29 PushButtonTextOn

Enables the text on a push button widget.

Definition:

```
#define PushButtonTextOn(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function enables the drawing of text on a push button widget. The display is not updated until the next paint request.

Returns:

None.

8.2.2.30 PushButtonTextOpaqueOff

Disables opaque text on a push button widget.

Definition:

```
#define PushButtonTextOpaqueOff(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function disables the use of opaque text on this push button. When not using opaque text, only the foreground pixels of the text are drawn on the screen, allowing the previously drawn pixels (such as the push button image) to show through the text.

Returns:

None.

8.2.2.31 PushButtonTextOpaqueOn

Enables opaque text on a push button widget.

Definition:

```
#define PushButtonTextOpaqueOn(pWidget)
```

Parameters:

pWidget is a pointer to the push button widget to modify.

Description:

This function enables the use of opaque text on this push button. When using opaque text, both the foreground and background pixels of the text are drawn on the screen, blocking out the previously drawn pixels.

Returns:

None.

8.2.2.32 PushButtonTextSet

Changes the text drawn on a push button widget.

Definition:

```
#define PushButtonTextSet(pWidget,
                          pcTxt)
```

Parameters:

pWidget is a pointer to the push button widget to be modified.

pcTxt is a pointer to the text to draw onto the push button.

Description:

This function changes the text that is drawn onto the push button. The display is not updated until the next paint request.

Returns:

None.

8.2.2.33 RectangularButton

Declares an initialized variable containing a rectangular push button widget data structure.

Definition:

```
#define RectangularButton(sName,
                          pParent,
                          pNext,
                          pChild,
                          pDisplay,
                          lX,
                          lY,
                          lWidth,
                          lHeight,
                          ulStyle,
                          ulFillColor,
                          ulPressFillColor,
                          ulOutlineColor,
                          ulTextColor,
                          pFont,
                          pcText,
                          pucImage,
                          pucPressImage,
                          usAutoRepeatDelay,
                          usAutoRepeatRate,
                          pfnOnClick)
```

Parameters:

sName is the name of the variable to be declared.
pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the push button.
lX is the X coordinate of the upper left corner of the push button.
lY is the Y coordinate of the upper left corner of the push button.
lWidth is the width of the push button.
lHeight is the height of the push button.
ulStyle is the style to be applied to the push button.
ulFillColor is the color used to fill in the push button.
ulPressFillColor is the color used to fill in the push button when it is pressed.
ulOutlineColor is the color used to outline the push button.
ulTextColor is the color used to draw text on the push button.
pFont is a pointer to the font to be used to draw text on the push button.
pcText is a pointer to the text to draw on this push button.
pucImage is a pointer to the image to draw on this push button.
pucPressImage is a pointer to the image to draw on this push button when it is pressed.
usAutoRepeatDelay is the delay before starting auto-repeat.
usAutoRepeatRate is the rate at which auto-repeat events are generated.
pfnOnClick is a pointer to the function that is called when the push button is pressed.

Description:

This macro provides an initialized rectangular push button widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls).

ulStyle is the logical OR of the following:

- **PB_STYLE_OUTLINE** to indicate that the push button should be outlined.
- **PB_STYLE_FILL** to indicate that the push button should be filled.
- **PB_STYLE_TEXT** to indicate that the push button should have text drawn on it (using *pFont* and *pcText*).
- **PB_STYLE_IMG** to indicate that the push button should have an image drawn on it (using *pucImage*).

- **PB_STYLE_TEXT_OPAQUE** to indicate that the push button text should be drawn opaque (in other words, drawing the background pixels).
- **PB_STYLE_AUTO_REPEAT** to indicate that auto-repeat should be used.

Returns:
Nothing; this is not a function.

8.2.2.34 RectangularButtonStruct

Declares an initialized rectangular push button widget data structure.

Definition:

```
#define RectangularButtonStruct(pParent,
                                pNext,
                                pChild,
                                pDisplay,
                                lX,
                                lY,
                                lWidth,
                                lHeight,
                                ulStyle,
                                ulFillColor,
                                ulPressFillColor,
                                ulOutlineColor,
                                ulTextColor,
                                pFont,
                                pcText,
                                pucImage,
                                pucPressImage,
                                usAutoRepeatDelay,
                                usAutoRepeatRate,
                                pfnOnClick)
```

Parameters:

pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the push button.
lX is the X coordinate of the upper left corner of the push button.
lY is the Y coordinate of the upper left corner of the push button.
lWidth is the width of the push button.
lHeight is the height of the push button.
ulStyle is the style to be applied to the push button.
ulFillColor is the color used to fill in the push button.
ulPressFillColor is the color used to fill in the push button when it is pressed.
ulOutlineColor is the color used to outline the push button.
ulTextColor is the color used to draw text on the push button.
pFont is a pointer to the font to be used to draw text on the push button.
pcText is a pointer to the text to draw on this push button.

pucImage is a pointer to the image to draw on this push button.
pucPressImage is a pointer to the image to draw on this push button when it is pressed.
usAutoRepeatDelay is the delay before starting auto-repeat.
usAutoRepeatRate is the rate at which auto-repeat events are generated.
pfnOnClick is a pointer to the function that is called when the push button is pressed.

Description:

This macro provides an initialized rectangular push button widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls). This must be assigned to a variable, such as:

```
tPushButtonWidget g_sPushButton = RectangularButtonStruct(...);
```

Or, in an array of variables:

```
tPushButtonWidget g_psPushButtons[] =
{
    RectangularButtonStruct(...),
    RectangularButtonStruct(...)
};
```

ulStyle is the logical OR of the following:

- **PB_STYLE_OUTLINE** to indicate that the push button should be outlined.
- **PB_STYLE_FILL** to indicate that the push button should be filled.
- **PB_STYLE_TEXT** to indicate that the push button should have text drawn on it (using *pFont* and *pcText*).
- **PB_STYLE_IMG** to indicate that the push button should have an image drawn on it (using *pucImage*).
- **PB_STYLE_TEXT_OPAQUE** to indicate that the push button text should be drawn opaque (in other words, drawing the background pixels).
- **PB_STYLE_AUTO_REPEAT** to indicate that auto-repeat should be used.

Returns:

Nothing; this is not a function.

8.2.3 Function Documentation

8.2.3.1 CircularButtonInit

Initializes a circular push button widget.

Prototype:

```
void
CircularButtonInit(tPushButtonWidget *pWidget,
                   const tDisplay *pDisplay,
                   long lX,
                   long lY,
                   long lR)
```

Parameters:

pWidget is a pointer to the push button widget to initialize.

pDisplay is a pointer to the display on which to draw the push button.
lX is the X coordinate of the upper left corner of the push button.
lY is the Y coordinate of the upper left corner of the push button.
lR is the radius of the push button.

Description:
This function initializes the provided push button widget so that it will be a circular push button.

Returns:
None.

8.2.3.2 CircularButtonMsgProc

Handles messages for a circular push button widget.

Prototype:
```
long
CircularButtonMsgProc(tWidget *pWidget,
                      unsigned long ulMsg,
                      unsigned long ulParam1,
                      unsigned long ulParam2)
```

Parameters:
pWidget is a pointer to the push button widget.
ulMsg is the message.
ulParam1 is the first parameter to the message.
ulParam2 is the second parameter to the message.

Description:
This function receives messages intended for this push button widget and processes them accordingly. The processing of the message varies based on the message in question.

Unrecognized messages are handled by calling WidgetDefaultMsgProc().

Returns:
Returns a value appropriate to the supplied message.

8.2.3.3 RectangularButtonInit

Initializes a rectangular push button widget.

Prototype:
```
void
RectangularButtonInit(tPushButtonWidget *pWidget,
                      const tDisplay *pDisplay,
                      long lX,
                      long lY,
                      long lWidth,
                      long lHeight)
```

Parameters:

pWidget is a pointer to the push button widget to initialize.

pDisplay is a pointer to the display on which to draw the push button.

lX is the X coordinate of the upper left corner of the push button.

lY is the Y coordinate of the upper left corner of the push button.

lWidth is the width of the push button.

lHeight is the height of the push button.

Description:

This function initializes the provided push button widget so that it will be a rectangular push button.

Returns:

None.

8.2.3.4 RectangularButtonMsgProc

Handles messages for a rectangular push button widget.

Prototype:

```
long
RectangularButtonMsgProc(tWidget *pWidget,
                         unsigned long ulMsg,
                         unsigned long ulParam1,
                         unsigned long ulParam2)
```

Parameters:

pWidget is a pointer to the push button widget.

ulMsg is the message.

ulParam1 is the first parameter to the message.

ulParam2 is the second parameter to the message.

Description:

This function receives messages intended for this push button widget and processes them accordingly. The processing of the message varies based on the message in question.

Unrecognized messages are handled by calling WidgetDefaultMsgProc().

Returns:

Returns a value appropriate to the supplied message.

9 Radio Button Widget

9.1 Introduction

The radio button widget provides a graphical element that can be grouped with other radio buttons to form a means of selecting one of many items. For example, three radio buttons can be grouped together to allow a selection between "low", "medium", and "high", where only one can be selected at a time. A radio button widget contains two graphical elements; the radio button itself (which is drawn as a circle that is either empty or contains a filled circle) and the radio button area around the radio button that visually indicates what the radio button controls.

When a radio button widget is drawn on the screen (via a **WIDGET_MSG_PAINT** request), the following sequence of drawing operations occurs:

- The radio button area is filled with the fill color if the radio button fill style is selected. The **RB_STYLE_FILL** flag enables filling of the radio button area.
- The radio button area is outlined with the outline color if the radio button outline style is selected. The **RB_STYLE_OUTLINE** flag enables outlining of the radio button area.
- The radio button is drawn, either empty if it is not selected or with a filled circle in the middle if it is selected.
- The radio button image is drawn next to the radio button if the radio button image style is selected. The **RB_STYLE_IMG** flag enables the image next to the radio button.
- The radio button text is drawn next to the radio button if the radio button text style is selected. The **RB_STYLE_TEXT** flag enables the text next to the radio button.

The steps are cumulative and any combination of these styles can be selected simultaneously. So, for example, the radio button can be filled, outlined, and then have a piece of text placed next to it.

A radio button works in cooperation with all the other radio buttons that have the same parent. Any number of radio buttons can be grouped together under a single parent to produce a one-of-many selection mechanism. Additionally, multiple groups of radio buttons can be grouped together under multiple parents to produce multiple, independent one-of-many selection mechanisms.

When a pointer down message is received within the extents of the radio button area, the behavior depends on the current state of the radio button. If the radio button is selected, then the pointer down message is ignored. If it is not selected, then the radio buttons in the group are unselected and this radio button is selected. An application callback is called when the state of a radio button changes, both when it is selected and unselected.

The container widget (described in chapter 7) blah.

9.2 Definitions

Data Structures

- tRadioButtonWidget

Defines

- RadioButton(sName, pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, usStyle, usCircleSize, ulFillColor,ulOutlineColor, ulTextColor, pFont, pcText, pucImage,pfnOnChange)
- RadioButtonCallbackSet(pWidget, pfnOnChg)
- RadioButtonCircleSizeSet(pWidget, usSize)
- RadioButtonFillColorSet(pWidget, ulColor)
- RadioButtonFillOff(pWidget)
- RadioButtonFillOn(pWidget)
- RadioButtonFontSet(pWidget, pFnt)
- RadioButtonImageOff(pWidget)
- RadioButtonImageOn(pWidget)
- RadioButtonImageSet(pWidget, pImg)
- RadioButtonOutlineColorSet(pWidget, ulColor)
- RadioButtonOutlineOff(pWidget)
- RadioButtonOutlineOn(pWidget)
- RadioButtonStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, usStyle, usCircleSize, ulFillColor,ulOutlineColor, ulTextColor, pFont, pcText,pucImage, pfnOnChange)
- RadioButtonTextColorSet(pWidget, ulColor)
- RadioButtonTextOff(pWidget)
- RadioButtonTextOn(pWidget)
- RadioButtonTextOpaqueOff(pWidget)
- RadioButtonTextOpaqueOn(pWidget)
- RadioButtonTextSet(pWidget, pcTxt)
- RB_STYLE_FILL
- RB_STYLE_IMG
- RB_STYLE_OUTLINE
- RB_STYLE_SELECTED
- RB_STYLE_TEXT
- RB_STYLE_TEXT_OPAQUE

Functions

- void RadioButtonInit (tRadioButtonWidget ∗pWidget, const tDisplay ∗pDisplay, long lX, long lY, long lWidth, long lHeight)
- long RadioButtonMsgProc (tWidget ∗pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)

9.2.1 Data Structure Documentation

9.2.1.1 tRadioButtonWidget

Definition:

```
typedef struct
{
    tWidget sBase;
    unsigned short usStyle;
    unsigned short usCircleSize;
    unsigned long ulFillColor;
    unsigned long ulOutlineColor;
    unsigned long ulTextColor;
    const tFont *pFont;
    const char *pcText;
    const unsigned char *pucImage;
    void (*pfnOnChange)(tWidget *pWidget,
                        unsigned long bSelected);
}
tRadioButtonWidget
```

Members:

sBase The generic widget information.

usStyle The style for this radio button. This is a set of flags defined by RB_STYLE_xxx.

usCircleSize The size of the radio button itself, not including the text and/or image that accompanies it (in other words, the size of the actual circle that is filled or unfilled).

ulFillColor The 24-bit RGB color used to fill this radio button, if RB_STYLE_FILL is selected, and to use as the background color if RB_STYLE_TEXT_OPAQUE is selected.

ulOutlineColor The 24-bit RGB color used to outline this radio button, if RB_STYLE_OUTLINE is selected.

ulTextColor The 24-bit RGB color used to draw text on this radio button, if RB_STYLE_TEXT is selected.

pFont The font used to draw the radio button text, if RB_STYLE_TEXT is selected.

pcText A pointer to the text to draw on this radio button, if RB_STYLE_TEXT is selected.

pucImage A pointer to the image to be drawn onto this radio button, if RB_STYLE_IMG is selected.

pfnOnChange A pointer to the function to be called when the radio button is pressed. This function is called when the state of the radio button is changed.

Description:

The structure that describes a radio button widget.

9.2.2 Define Documentation

9.2.2.1 RadioButton

Declares an initialized variable containing a radio button widget data structure.

Definition:

```
#define RadioButton(sName,
                    pParent,
                    pNext,
                    pChild,
                    pDisplay,
                    lX,
                    lY,
                    lWidth,
                    lHeight,
                    usStyle,
                    usCircleSize,
                    ulFillColor,
                    ulOutlineColor,
                    ulTextColor,
                    pFont,
                    pcText,
                    pucImage,
                    pfnOnChange)
```

Parameters:

sName is the name of the variable to be declared.
pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the radio button.
lX is the X coordinate of the upper left corner of the radio button.
lY is the Y coordinate of the upper left corner of the radio button.
lWidth is the width of the radio button.
lHeight is the height of the radio button.
usStyle is the style to be applied to this radio button.
usCircleSize is the size of the circle that is filled.
ulFillColor is the color used to fill in the radio button.
ulOutlineColor is the color used to outline the radio button.
ulTextColor is the color used to draw text on the radio button.
pFont is a pointer to the font to be used to draw text on the radio button.
pcText is a pointer to the text to draw on this radio button.
pucImage is a pointer to the image to draw on this radio button.
pfnOnChange is a pointer to the function that is called when the radio button is pressed.

Description:

This macro provides an initialized radio button widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls).

usStyle is the logical OR of the following:

- **RB_STYLE_OUTLINE** to indicate that the radio button should be outlined.
- **RB_STYLE_FILL** to indicate that the radio button should be filled.
- **RB_STYLE_TEXT** to indicate that the radio button should have text drawn on it (using *pFont* and *pcText*).

- **RB_STYLE_IMG** to indicate that the radio button should have an image drawn on it (using *pucImage*).
- **RB_STYLE_TEXT_OPAQUE** to indicate that the radio button text should be drawn opaque (in other words, drawing the background pixels).
- **RB_STYLE_SELECTED** to indicate that the radio button is selected.

Returns:
Nothing; this is not a function.

9.2.2.2 RadioButtonCallbackSet

Sets the function to call when this radio button widget is toggled.

Definition:
```
#define RadioButtonCallbackSet(pWidget,
                               pfnOnChg)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.
pfnOnChg is a pointer to the function to call.

Description:
This function sets the function to be called when this radio button is toggled.

Returns:
None.

9.2.2.3 RadioButtonCircleSizeSet

Sets size of the circle to be filled.

Definition:
```
#define RadioButtonCircleSizeSet(pWidget,
                                 usSize)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.
usSize is the size of the circle, in pixels.

Description:
This function sets the size of the circle that is drawn as part of the radio button.

Returns:
None.

9.2.2.4 RadioButtonFillColorSet

Sets the fill color of a radio button widget.

Definition:
```
#define RadioButtonFillColorSet(pWidget,
                                ulColor)
```

Parameters:
pWidget is a pointer to the radio button widget to be modified.
ulColor is the 24-bit RGB color to use to fill the radio button.

Description:
This function changes the color used to fill the radio button on the display. The display is not updated until the next paint request.

Returns:
None.

9.2.2.5 RadioButtonFillOff

Disables filling of a radio button widget.

Definition:
```
#define RadioButtonFillOff(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function disables the filling of a radio button widget. The display is not updated until the next paint request.

Returns:
None.

9.2.2.6 RadioButtonFillOn

Enables filling of a radio button widget.

Definition:
```
#define RadioButtonFillOn(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function enables the filling of a radio button widget. The display is not updated until the next paint request.

Returns:
None.

9.2.2.7 RadioButtonFontSet

Sets the font for a radio button widget.

Definition:

```
#define RadioButtonFontSet(pWidget,
                           pFnt)
```

Parameters:

pWidget is a pointer to the radio button widget to modify.

pFnt is a pointer to the font to use to draw text on the radio button.

Description:

This function changes the font used to draw text on the radio button. The display is not updated until the next paint request.

Returns:

None.

9.2.2.8 RadioButtonImageOff

Disables the image on a radio button widget.

Definition:

```
#define RadioButtonImageOff(pWidget)
```

Parameters:

pWidget is a pointer to the radio button widget to modify.

Description:

This function disables the drawing of an image on a radio button widget. The display is not updated until the next paint request.

Returns:

None.

9.2.2.9 RadioButtonImageOn

Enables the image on a radio button widget.

Definition:

```
#define RadioButtonImageOn(pWidget)
```

Parameters:

pWidget is a pointer to the radio button widget to modify.

Description:

This function enables the drawing of an image on a radio button widget. The display is not updated until the next paint request.

Returns:

None.

9.2.2.10 RadioButtonImageSet

Changes the image drawn on a radio button widget.

Definition:
```
#define RadioButtonImageSet(pWidget,
                            pImg)
```

Parameters:
pWidget is a pointer to the radio button widget to be modified.
pImg is a pointer to the image to draw onto the radio button.

Description:
This function changes the image that is drawn onto the radio button. The display is not updated until the next paint request.

Returns:
None.

9.2.2.11 RadioButtonOutlineColorSet

Sets the outline color of a radio button widget.

Definition:
```
#define RadioButtonOutlineColorSet(pWidget,
                                   ulColor)
```

Parameters:
pWidget is a pointer to the radio button widget to be modified.
ulColor is the 24-bit RGB color to use to outline the radio button.

Description:
This function changes the color used to outline the radio button on the display. The display is not updated until the next paint request.

Returns:
None.

9.2.2.12 RadioButtonOutlineOff

Disables outlining of a radio button widget.

Definition:
```
#define RadioButtonOutlineOff(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function disables the outlining of a radio button widget. The display is not updated until the next paint request.

Returns:
None.

9.2.2.13 RadioButtonOutlineOn

Enables outlining of a radio button widget.

Definition:
```
#define RadioButtonOutlineOn(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function enables the outlining of a radio button widget. The display is not updated until the next paint request.

Returns:
None.

9.2.2.14 RadioButtonStruct

Declares an initialized radio button widget data structure.

Definition:
```
#define RadioButtonStruct(pParent,
                          pNext,
                          pChild,
                          pDisplay,
                          lX,
                          lY,
                          lWidth,
                          lHeight,
                          usStyle,
                          usCircleSize,
                          ulFillColor,
                          ulOutlineColor,
                          ulTextColor,
                          pFont,
                          pcText,
                          pucImage,
                          pfnOnChange)
```

Parameters:
pParent is a pointer to the parent widget.
pNext is a pointer to the sibling widget.
pChild is a pointer to the first child widget.
pDisplay is a pointer to the display on which to draw the radio button.
lX is the X coordinate of the upper left corner of the radio button.
lY is the Y coordinate of the upper left corner of the radio button.

lWidth is the width of the radio button.
lHeight is the height of the radio button.
usStyle is the style to be applied to this radio button.
usCircleSize is the size of the circle that is filled.
ulFillColor is the color used to fill in the radio button.
ulOutlineColor is the color used to outline the radio button.
ulTextColor is the color used to draw text on the radio button.
pFont is a pointer to the font to be used to draw text on the radio button.
pcText is a pointer to the text to draw on this radio button.
pucImage is a pointer to the image to draw on this radio button.
pfnOnChange is a pointer to the function that is called when the radio button is pressed.

Description:

This macro provides an initialized radio button widget data structure, which can be used to construct the widget tree at compile time in global variables (as opposed to run-time via function calls). This must be assigned to a variable, such as:

```
tRadioButtonWidget g_sRadioButton = RadioButtonStruct(...);
```

Or, in an array of variables:

```
tRadioButtonWidget g_psRadioButtons[] =
{
    RadioButtonStruct(...),
    RadioButtonStruct(...)
};
```

usStyle is the logical OR of the following:

- **RB_STYLE_OUTLINE** to indicate that the radio button should be outlined.
- **RB_STYLE_FILL** to indicate that the radio button should be filled.
- **RB_STYLE_TEXT** to indicate that the radio button should have text drawn on it (using *pFont* and *pcText*).
- **RB_STYLE_IMG** to indicate that the radio button should have an image drawn on it (using *pucImage*).
- **RB_STYLE_TEXT_OPAQUE** to indicate that the radio button text should be drawn opaque (in other words, drawing the background pixels).
- **RB_STYLE_SELECTED** to indicate that the radio button is selected.

Returns:

Nothing; this is not a function.

9.2.2.15 RadioButtonTextColorSet

Sets the text color of a radio button widget.

Definition:

```
#define RadioButtonTextColorSet(pWidget,
                                ulColor)
```

Parameters:

pWidget is a pointer to the radio button widget to be modified.

ulColor is the 24-bit RGB color to use to draw text on the radio button.

Description:
This function changes the color used to draw text on the radio button on the display. The display is not updated until the next paint request.

Returns:
None.

9.2.2.16 RadioButtonTextOff

Disables the text on a radio button widget.

Definition:
```
#define RadioButtonTextOff(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function disables the drawing of text on a radio button widget. The display is not updated until the next paint request.

Returns:
None.

9.2.2.17 RadioButtonTextOn

Enables the text on a radio button widget.

Definition:
```
#define RadioButtonTextOn(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function enables the drawing of text on a radio button widget. The display is not updated until the next paint request.

Returns:
None.

9.2.2.18 RadioButtonTextOpaqueOff

Disables opaque text on a radio button widget.

Definition:
```
#define RadioButtonTextOpaqueOff(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function disables the use of opaque text on this radio button. When not using opaque text, only the foreground pixels of the text are drawn on the screen, allowing the previously drawn pixels (such as the radio button image) to show through the text.

Returns:
None.

9.2.2.19 RadioButtonTextOpaqueOn

Enables opaque text on a radio button widget.

Definition:
```
#define RadioButtonTextOpaqueOn(pWidget)
```

Parameters:
pWidget is a pointer to the radio button widget to modify.

Description:
This function enables the use of opaque text on this radio button. When using opaque text, both the foreground and background pixels of the text are drawn on the screen, blocking out the previously drawn pixels.

Returns:
None.

9.2.2.20 RadioButtonTextSet

Changes the text drawn on a radio button widget.

Definition:
```
#define RadioButtonTextSet(pWidget,
                           pcTxt)
```

Parameters:
pWidget is a pointer to the radio button widget to be modified.
pcTxt is a pointer to the text to draw onto the radio button.

Description:
This function changes the text that is drawn onto the radio button. The display is not updated until the next paint request.

Returns:
None.

9.2.2.21 RB_STYLE_FILL

Definition:

```
#define RB_STYLE_FILL
```

Description:

This flag indicates that the radio button should be filled.

9.2.2.22 RB_STYLE_IMG

Definition:

```
#define RB_STYLE_IMG
```

Description:

This flag indicates that the radio button should have an image drawn on it.

9.2.2.23 RB_STYLE_OUTLINE

Definition:

```
#define RB_STYLE_OUTLINE
```

Description:

This flag indicates that the radio button should be outlined.

9.2.2.24 RB_STYLE_SELECTED

Definition:

```
#define RB_STYLE_SELECTED
```

Description:

This flag indicates that the radio button is selected.

9.2.2.25 RB_STYLE_TEXT

Definition:

```
#define RB_STYLE_TEXT
```

Description:

This flag indicates that the radio button should have text drawn on it.

9.2.2.26 RB_STYLE_TEXT_OPAQUE

Definition:

```
#define RB_STYLE_TEXT_OPAQUE
```

Description:

This flag indicates that the radio button text should be drawn opaque (in other words, drawing the background pixels as well as the foreground pixels).

9.2.3 Function Documentation

9.2.3.1 RadioButtonInit

Initializes a radio button widget.

Prototype:
```
void
RadioButtonInit(tRadioButtonWidget *pWidget,
                const tDisplay *pDisplay,
                long lX,
                long lY,
                long lWidth,
                long lHeight)
```

Parameters:

pWidget is a pointer to the radio button widget to initialize.
pDisplay is a pointer to the display on which to draw the push button.
lX is the X coordinate of the upper left corner of the radio button.
lY is the Y coordinate of the upper left corner of the radio button.
lWidth is the width of the radio button.
lHeight is the height of the radio button.

Description:

This function initializes the provided radio button widget.

Returns:

None.

9.2.3.2 RadioButtonMsgProc

Handles messages for a radio button widget.

Prototype:
```
long
RadioButtonMsgProc(tWidget *pWidget,
                   unsigned long ulMsg,
                   unsigned long ulParam1,
                   unsigned long ulParam2)
```

Parameters:

pWidget is a pointer to the radio button widget.
ulMsg is the message.
ulParam1 is the first parameter to the message.
ulParam2 is the second parameter to the message.

Description:

This function receives messages intended for this radio button widget and processes them accordingly. The processing of the message varies based on the message in question.

Unrecognized messages are handled by calling WidgetDefaultMsgProc().

Returns:

Returns a value appropriate to the supplied message.

10 Utilities

10.1 Introduction

There are several utility applications that can be used to produce the data structures required by the graphics library for fonts and images since trying to produce these structures by hand would be a difficult process. The use of these utilities is not required in order to use the graphics library, though they do make it much easier to use.

10.2 ftrasterize

The ftrasterize utility uses the FreeType font rendering package to convert a font into the format that is recognized by the graphics library. Any font that is recognized by FreeType can be used, which include TrueType®, OpenType®, PostScript® Type 1, and Windows® FNT fonts. A complete list of supported font formats can be found on the FreeType web site at `http://www.freetype.org.`

FreeType is used to render the glyphs of a font at a specific size in monochrome, using the result as the bitmap images for the font. These bitmaps are optionally compressed, and the results are written as a C source file that provides a tFont structure describing the font.

The application is run from the command line, and its usage is as follows:

```
ftrasterize [-b] [-f <filename>] [-i] [-s <size>] <font>
```

Where the arguments mean:

`-b`	Specifies that this is a bold font. This does not affect the rendering of the font, it only changes the name of the file and the name of the font structure that are produced.
`-f <filename>`	Specifies the base name for this font, which is used to create the output file name and the name of the font structure. The default value is "font" if not specified.
`-i`	Specifies that this is an italic font. This does not affect the rendering of the font, it only changes the name of the file and the name of the font structure that are produced.
`-s <size>`	Specifies the size of this font, in points. The default value is 20 if not specified.
`<font>`	Specifies the name of the input font file to be processed.

For example, to produce a 24 point font called test from test.ttf, use the following:

```
ftrasterize -f test -s 24 test.ttf
```

The result will be written to fonttest24.c, and will contain a structure called g_sFontTest24 that describes the font.

This application is located in `grlib/ftrasterize`.

10.3 lmi-button

The lmi-button script is a script-fu plugin for GIMP (http://www.gimp.org) that produces push button images that can be used by the push button widget. When installed into `${HOME}/.gimp-2.4/scripts`, this will be available under Xtns->Buttons->LMI Button. When run, a dialog will be displayed allowing the width and height of the button, the radius of the corners, the thickness of the 3D effect, the color of the button, and the pressed state of the button to be selected. Once the desired configuration is selected, pressing OK will create the push button image in a new GIMP image. The image should be saved as a raw PPM file so that it can be converted to a C array by pnmtoc.

This script is provided as a convenience to easily produce a particular push button appearance; the push button images can be of any desired appearance.

This script is located in `grlib/pnmtoc/lmi-button.scm`.

10.4 pnmtoc

The pnmtoc utility converts a NetPBM image file into the format that is recognized by the graphics library. The input image must be in the raw PPM format (in other words, with the `P6` tag). The NetPBM image format can be produced using GIMP, NetPBM (http://netpbm.sourceforge.net), ImageMagick (http://www.imagemagick.org), or numerous other open source and proprietary image manipulation packages.

The application is run from the command line, and its usage is as follows:

```
pnmtoc [-c] <file>
```

Where the arguments mean:

`-c`	Specifies that the image should be compressed. Compression is bypassed if it would result in a larger C array.
`<file>`	Specifies the input image file.

The resulting C image array definition is written to standard output; this follows the convention of the NetPBM toolkit after which the application was modeled (both in behavior and naming). The output should be redirected into a file so that it can then be used by the application.

For example,to produce a compressed image in foo.c from foo.ppm, use the following:

```
pnmtoc -c foo.ppm > foo.c
```

This will result in an array called g_pucImage that contains the image data from foo.ppm. If foo.ppm contains only two colors, the 1 BPP image format is used; if it contains 16 or less colors, the 4 BPP image format is used; if it contains 256 or less colors, the 8 BPP image format used; and if it contains more than 256 colors an error is generated.

The number of colors in an image can be reduced to the desired number using any number of techniques. The easiest is to use the Posterize effect (Colors->Posterize) in GIMP, which selects the N best colors for the image and translates every pixel to one of those N colors. For example, selecting N as 16 will result in a 4 BPP image.

This application is located in `grlib/pnmtoc`.

Company Information

Founded in 2004, Luminary Micro, Inc. designs, markets, and sells ARM Cortex-M3-based microcontrollers (MCUs). Austin, Texas-based Luminary Micro is the lead partner for the Cortex-M3 processor, delivering the world's first silicon implementation of the Cortex-M3 processor. Luminary Micro's introduction of the Stellaris family of products provides 32-bit performance for the same price as current 8- and 16-bit microcontroller designs. With entry-level pricing at $1.00 for an ARM technology-based MCU, Luminary Micro's Stellaris product line allows for standardization that eliminates future architectural upgrades or software tool changes.

Luminary Micro, Inc.
108 Wild Basin, Suite 350
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com
sales@luminarymicro.com

Support Information

For support on Luminary Micro products, contact:

support@luminarymicro.com
+1-512-279-8800, ext 3