Stellaris® Graphics Library

USER'S GUIDE

TEXAS INSTRUMENTS

Copyright

Revision Information

This is version 7611 of this document, last updated on July 02, 2011.

Table of Contents

1 Introduction

The Texas Instruments® 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.
- They can be built with more than one tool chain.

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 particular 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.

The following tool chains are supported:

- Keil™ RealView® Microcontroller Development Kit
- CodeSourcery Sourcery G++ for Stellaris EABI

- IAR Embedded Workbench®
- Code Red Technologies tools
- Texas Instruments Code Composer Studio™

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.

`Makefile`	The rules for building the graphics library.
`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.
`ccs/`	The directory containing the Code Composer Studio project files.
`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. A list of the fonts, along with a sample string rendered in each font, is provided in chapter 15.
`ftrasterize/`	The source code for the ftrasterize program, which is described in chapter 13.
`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.ewd`	The IAR Embedded Workbench project options file used for building the graphics library.
`grlib.ewp`	The IAR Embedded Workbench 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 13.
`pushbutton.c`	The source code for the push button widget, which is described in chapter 10.
`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 11.
`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.
`slider.c`	The source code for the slider widget, which is described in chapter 12.
`slider.h`	The header containing prototypes for the slider widget.
`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 containing 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 function 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 set of predefined colors are provided in `grlib/grlib.h` that can be used if desired; a list of these colors and a corresponding color swatch can be found in chapter 14.

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 T$_E$X 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. A list of these fonts and a sample string rendered in each font can be found in chapter 15.

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

The code for the graphics primitives are contained in `grlib/circle.c`, `grlib/context.c`, `grlib/image.c`, `grlib/line.c`, `grlib/offscr1bpp.c`, `grlib/offscr4bpp.c`, `grlib/offscr8bpp.c`, `grlib/rectangle.c`, and `grlib/string.c`, with `grlib/grlib.h` containing the API definitions for use by applications.

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 other 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.

A utility (pnmtoc) is provided to assist in converting images into this format; it is documented in chapter 13.

3.1.4   Font Format

Font data is stored with a scheme that 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.

Two structures are used to describe a font to the graphics library functions. The `tFont` structure allows encoding of fonts containing the printable subset of the 7-bit ASCII encoding, characters 32 (space) to 126 (tilde). The `tFontEx` structure allows encoding of any arbitrary subset of characters 0 through 255 and may be used to support any ISO8859 variant, allowing the use of, for example Western European accented characters in addition to the basic Latin alphabet. This format is also useful when only a subset of characters from a given font are needed. For example, an application requiring only digits from a 44 point font can encode only these 10 characters using a `tFontEx` structure and save the memory that would be required if using `tFont` instead.

A utility (ftrasterize) is provided to assist in converting fonts into these formats; it is documented in chapter 13.

3.2   Definitions

Data Structures

- tContext
- tDisplay
- tFont
- tFontEx
- 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_EX_MARKER
- FONT_FMT_EX_PIXEL_RLE

- FONT_FMT_EX_UNCOMPRESSED
- 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)
- GrRectContainsPoint(pRect, 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)
- long GrRectIntersectGet (tRectangle ∗psRect1, tRectangle ∗psRect2, tRectangle ∗psIntersect)
- long GrRectOverlapCheck (tRectangle ∗psRect1, tRectangle ∗psRect2)
- void GrStringDraw (const tContext ∗pContext, const char ∗pcString, long lLength, long lX, long lY, unsigned long bOpaque)
- unsigned long GrStringGet (long lIndex, char ∗pcData, unsigned long ulSize)
- unsigned long GrStringLanguageSet (unsigned short usLangID)
- void GrStringTableSet (const void ∗pvTable)
- 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   tFontEx

Definition:

```
typedef struct
{
    unsigned char ucFormat;
    unsigned char ucMaxWidth;
    unsigned char ucHeight;
    unsigned char ucBaseline;
    unsigned char ucFirst;
    unsigned char ucLast;
    const unsigned short *pusOffset;
    const unsigned char *pucData;
}
tFontEx
```

Members:

> ***ucFormat*** The format of the font. Can be one of FONT_FMT_EX_UNCOMPRESSED or FONT_FMT_EX_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.

> ***ucFirst*** The codepoint number representing the first character encoded in the font.

> ***ucLast*** The codepoint number representing the last character encoded in the font.

> ***pusOffset*** A pointer to a table containing the offset within pucData to the data for each character in the font.

> ***pucData*** A pointer to the data for the font.

Description:

> This is a newer version of the structure which describes a font used for drawing text onto the screen. This variant allows a font to contain an arbitrary, contiguous block of codepoints from the 256 basic characters in an ISO8859-n font and allows support for accented characters in Western European languages and any left-to-right typeface supported by an ISO8859 variant. Fonts encoded in this format may be used interchangeably with the original fonts merely by casting the structure pointer when calling any function or macro which expects a font pointer as a parameter.

3.2.1.5   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_EX_MARKER

Definition:
```
#define FONT_EX_MARKER
```

Description:
A marker used in the ucFormat field of a font to indicates that the font data is stored using the new tFontEx structure.

3.2.2.11 FONT_FMT_EX_PIXEL_RLE

Definition:
```
#define FONT_FMT_EX_PIXEL_RLE
```

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

3.2.2.12 FONT_FMT_EX_UNCOMPRESSED

Definition:
```
#define FONT_FMT_EX_UNCOMPRESSED
```

Description:
Indicates that the font data is stored in an uncompressed format and uses the tFontEx structure format.

3.2.2.13 FONT_FMT_PIXEL_RLE

Definition:
```
#define FONT_FMT_PIXEL_RLE
```

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

3.2.2.14 FONT_FMT_UNCOMPRESSED

Definition:
```
#define FONT_FMT_UNCOMPRESSED
```

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

3.2.2.15 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.

3.2.2.16   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.

3.2.2.17   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.

3.2.2.18   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.

3.2.2.19 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. If a tFontEx type font is to be used, cast its pointer to a pFont pointer before passing it as the pFnt parameter.

Returns:
>   None.

3.2.2.20 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.

3.2.2.21 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 draw-
> ing context, using a color that has been previously translated to a driver-specific color (for
> example, via DpyColorTranslate()).

Returns:
> None.

3.2.2.22 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.

3.2.2.23 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.

3.2.2.24 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.

3.2.2.25 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.

3.2.2.26 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.

3.2.2.27   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.

3.2.2.28   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.

3.2.2.29   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.

3.2.2.30 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.

3.2.2.31 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.

3.2.2.32 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.

3.2.2.33 GrRectContainsPoint

Determines if a point lies within a given rectangle.

Definition:

```
#define GrRectContainsPoint(pRect,
                            lX,
                            lY)
```

Parameters:

>**pRect** is a pointer to the rectangle which the point is to be checked against.
>
>**lX** is the X coordinate of the point to be checked.
>
>**lY** is the Y coordinate of the point to be checked.

Description:

>This function determines whether point (lX, lY) lies within the rectangle described by *pRect*.

Returns:

>Returns 1 if the point is within the rectangle or 0 otherwise.

3.2.2.34 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.

3.2.2.35 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.

3.2.2.36 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.

3.2.2.37 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.

3.2.2.38 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.

3.2.2.39 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.

3.2.2.40 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.

3.2.2.41 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.

3.2.2.42 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.

3.2.2.43 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 GrRectIntersectGet

Determines the intersection of two rectangles.

Prototype:
```
long
GrRectIntersectGet(tRectangle *psRect1,
                   tRectangle *psRect2,
                   tRectangle *psIntersect)
```

Parameters:
psRect1 is a pointer to the first rectangle.

psRect2 is a pointer to the second rectangle.

psIntersect is a pointer to a rectangle which will be written with the intersection of *psRect1* and *psRect2*.

Description:
This function determines if two rectangles overlap and, if they do, calculates the rectangle representing their intersection. If the rectangles do not overlap, 0 is returned and *psIntersect* is not written.

Returns:
Returns 1 if there is an overlap or 0 if not.

3.2.3.17 GrRectOverlapCheck

Determines if two rectangles overlap.

Prototype:
```
long
GrRectOverlapCheck(tRectangle *psRect1,
                   tRectangle *psRect2)
```

Parameters:
> ***psRect1*** is a pointer to the first rectangle.
>
> ***psRect2*** is a pointer to the second rectangle.

Description:
> This function determines whether two rectangles overlap. It assumes that rectangles *psRect1* and *psRect2* are valid with *sXMin* < *sXMax* and *sYMin* < *sYMax*.

Returns:
> Returns 1 if there is an overlap or 0 if not.

3.2.3.18 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.

3.2.3.19 GrStringGet

This function returns a string from the current string table.

Prototype:
```
unsigned long
GrStringGet(long lIndex,
            char *pcData,
            unsigned long ulSize)
```

Parameters:
>   ***lIndex*** is the index of the string to retrieve.
>   ***pcData*** is the pointer to the buffer to store the string into.
>   ***ulSize*** is the size of the buffer provided by pcData.

Description:
>   This function will return a string from the string table in the language set by the GrStringLanguageSet() function. The value passed in *lIndex* parameter is the string that is being requested and will be returned in the buffer provided in the *pcData* parameter. The amount of data returned will be limited by the ulSize parameter.

Returns:
>   Returns the number of valid bytes returned in the *pcData* buffer.

3.2.3.20 GrStringLanguageSet

This function sets the current language for strings returned by the GrStringGet() function.

Prototype:
```
unsigned long
GrStringLanguageSet(unsigned short usLangID)
```

Parameters:
>   ***usLangID*** is one of the language identifiers provided in the string table.

Description:
>   This function is used to set the language identifier for the strings returned by the GrStringGet() function. The *usLangID* parameter should match one of the identifiers that was included in the string table. These are provided in a header file in the graphics library and must match the values that were passed through the sting compression utility.

Returns:
>   This function returns 0 if the language was not found and a non-zero value if the laguage was found.

3.2.3.21 GrStringTableSet

This function sets the location of the current string table.

Prototype:
```
void
GrStringTableSet(const void *pvTable)
```

Parameters:
>   ***pvTable*** is a pointer to a string table that was generated by the string compression utility.

Description:
>   This function is used to set the string table to use for strings in an application. This string table is created by the string compression utility. This function is used to swap out multiple string tables if the application requires more than one table. It does not allow using more than one string table at a time.

Returns:
>   None.

3.2.3.22   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.

The code for the widget framework is contained in `grlib/widget.c`, with `grlib/widget.h` containing the API definitions for use by applications.

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)
- unsigned long WidgetMutexGet (unsigned char ∗pcMutex)
- void WidgetMutexInit (unsigned char ∗pcMutex)
- void WidgetMutexPut (unsigned char ∗pcMutex)
- 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. To add to the root of the tree set this parameter to **WIDGET_ROOT**.
> **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 either the queue is full or another context is currently adding a message to the queue.

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 **WID-
> GET_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 mes-
> sage, 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.
>
> ***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   WidgetMutexGet

Attempts to acquire a mutex.

Prototype:
```
unsigned long
WidgetMutexGet(unsigned char *pcMutex)
```

Parameters:
pcMutex is a pointer to mutex that is to be acquired.

Description:
This function attempts to acquire a mutual exclusion semaphore (mutex) on behalf of the caller. If the mutex is not already held, 0 is returned to indicate that the caller may safely access whichever resource the mutex is protecting. If the mutex is already held, 1 is returned and the caller must not access the shared resource.

When access to the shared resource is complete, the mutex owner should call WidgetMutexPut() to release the mutex and relinquish ownership of the shared resource.

Returns:
Returns 0 if the mutex is acquired successfully or 1 if it is already held by another caller.

4.2.3.8   WidgetMutexInit

Initializes a mutex to the unowned state.

Prototype:
```
void
WidgetMutexInit(unsigned char *pcMutex)
```

Parameters:
pcMutex is a pointer to mutex that is to be initialized.

Description:
This function initializes a mutual exclusion semaphore (mutex) to its unowned state in preparation for use with WidgetMutexGet() and WidgetMutexPut(). A mutex is a two state object typically used to serialize access to a shared resource. An application will call WidgetMutexGet() to request ownership of the mutex. If ownership is granted, the caller may safely access the resource then release the mutex using WidgetMutexPut() once it is finished. If ownership is not

granted, the caller knows that some other context is currently modifying the shared resource and it must not access the resource at that time.

Note that this function must not be called if the mutex passed in *pcMutex* is already in use since this will have the effect of releasing the lock even if some caller currently owns it.

Returns:
None.

4.2.3.9 WidgetMutexPut

Release a mutex.

Prototype:
```
void
WidgetMutexPut(unsigned char *pcMutex)
```

Parameters:
pcMutex is a pointer to mutex that is to be released.

Description:
This function releases a mutual exclusion semaphore (mutex), leaving it in the unowned state.

Returns:
None.

4.2.3.10 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.

4.2.3.11 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 tree, allowing removal of an entire heirarchy all at once (for example, removing an entire screen from the widget tree).

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 within it, 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 onto canvas if the canvas text style is selected. The **CANVAS_STYLE_TEXT** flag enables the text on the canvas and flags **CANVAS_STYLE_TEXT_LEFT**, **CANVAS_STYLE_TEXT_RIGHT**, **CANVAS_STYLE_TEXT_TOP** and **CANVAS_STYLE_TEXT_BOTTOM** control alignment within the widget. If no alignment style is given for a particular axis, the text is centered on that axis.

- 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_BOTTOM
- CANVAS_STYLE_TEXT_HCENTER
- CANVAS_STYLE_TEXT_LEFT
- CANVAS_STYLE_TEXT_OPAQUE
- CANVAS_STYLE_TEXT_RIGHT
- CANVAS_STYLE_TEXT_TOP
- CANVAS_STYLE_TEXT_VCENTER
- 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)
- CanvasTextAlignment(pWidget, ulAlign)
- 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   Detailed Description

The code for this widget is contained in `grlib/canvas.c`, with `grlib/canvas.h` containing the API definitions for use by applications.

5.2.2   Data Structure Documentation

5.2.2.1   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 CAN-VAS_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 se-lected.

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 can-vas, if CANVAS_STYLE_APP_DRAWN is selected.

Description:

The structure that describes a canvas widget.

5.2.3   Define Documentation

5.2.3.1   Canvas

Declares an initialized variable containing a canvas widget data structure.

Definition:
```
#define Canvas(sName,
                pParent,
                pNext,
                pChild,
                pDisplay,
                lX,
                lY,
                lWidth,
                lHeight,
                ulStyle,
                ulFillColor,
                ulOutlineColor,
                ulTextColor,
                pFont,
                pcText,
                pucImage,
                pfnOnPaint)
```

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.

Description:

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).
- **CANVAS_STYLE_TEXT_LEFT** to indicate that the canvas text should be left aligned within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_HCENTER** to indicate that the canvas text should be horizontally centered within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_RIGHT** to indicate that the canvas text should be right aligned within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_TOP** to indicate that the canvas text should be top aligned within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_VCENTER** to indicate that the canvas text should be vertically centered within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_BOTTOM** to indicate that the canvas text should be bottom aligned within the widget bounding rectangle.

Returns:

Nothing; this is not a function.

5.2.3.2   CANVAS_STYLE_APP_DRAWN

Definition:

```
#define CANVAS_STYLE_APP_DRAWN
```

Description:

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

5.2.3.3   CANVAS_STYLE_FILL

Definition:

```
#define CANVAS_STYLE_FILL
```

Description:

This flag indicates that the canvas should be filled.

5.2.3.4   CANVAS_STYLE_IMG

Definition:

```
#define CANVAS_STYLE_IMG
```

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

5.2.3.5 CANVAS_STYLE_OUTLINE

Definition:
```
#define CANVAS_STYLE_OUTLINE
```

Description:
This flag indicates that the canvas should be outlined.

5.2.3.6 CANVAS_STYLE_TEXT

Definition:
```
#define CANVAS_STYLE_TEXT
```

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

5.2.3.7 CANVAS_STYLE_TEXT_BOTTOM

Definition:
```
#define CANVAS_STYLE_TEXT_BOTTOM
```

Description:
This flag indicates that canvas text should be bottom-aligned. By default, text is centered in both X and Y within the canvas bounding rectangle.

5.2.3.8 CANVAS_STYLE_TEXT_HCENTER

Definition:
```
#define CANVAS_STYLE_TEXT_HCENTER
```

Description:
This flag indicates that canvas text should be centered horizontally. By default, text is centered in both X and Y within the canvas bounding rectangle.

5.2.3.9 CANVAS_STYLE_TEXT_LEFT

Definition:
```
#define CANVAS_STYLE_TEXT_LEFT
```

Description:
This flag indicates that canvas text should be left-aligned. By default, text is centered in both X and Y within the canvas bounding rectangle.

5.2.3.10 CANVAS_STYLE_TEXT_OPAQUE

Definition:
```
#define CANVAS_STYLE_TEXT_OPAQUE
```

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

5.2.3.11 CANVAS_STYLE_TEXT_RIGHT

Definition:
```
#define CANVAS_STYLE_TEXT_RIGHT
```

Description:
This flag indicates that canvas text should be right-aligned. By default, text is centered in both X and Y within the canvas bounding rectangle.

5.2.3.12 CANVAS_STYLE_TEXT_TOP

Definition:
```
#define CANVAS_STYLE_TEXT_TOP
```

Description:
This flag indicates that canvas text should be top-aligned. By default, text is centered in both X and Y within the canvas bounding rectangle.

5.2.3.13 CANVAS_STYLE_TEXT_VCENTER

Definition:
```
#define CANVAS_STYLE_TEXT_VCENTER
```

Description:
This flag indicates that canvas text should be centered vertically. By default, text is centered in both X and Y within the canvas bounding rectangle.

5.2.3.14 CanvasAppDrawnOff

Disables application drawing of a canvas widget.

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

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

Description:
> 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.

5.2.3.15 CanvasAppDrawnOn

Enables application drawing of a canvas widget.

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

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

Description:
> 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.

5.2.3.16 CanvasCallbackSet

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

Definition:
```
#define CanvasCallbackSet(pWidget,
                          pfnOnPnt)
```

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

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

Returns:
> None.

5.2.3.17 CanvasFillColorSet

Sets the fill color of a canvas widget.

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

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.

Description:
> 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.

5.2.3.18 CanvasFillOff

Disables filling of a canvas widget.

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

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

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

Returns:
> None.

5.2.3.19 CanvasFillOn

Enables filling of a canvas widget.

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

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

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

Returns:
> None.

5.2.3.20 CanvasFontSet

Sets the font for a canvas widget.

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

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.

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

Returns:
> None.

5.2.3.21 CanvasImageOff

Disables the image on a canvas widget.

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

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

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

Returns:
> None.

5.2.3.22 CanvasImageOn

Enables the image on a canvas widget.

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

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

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

Returns:
> None.

5.2.3.23 CanvasImageSet

Changes the image drawn on a canvas widget.

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

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

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

Returns:
None.

5.2.3.24 CanvasOutlineColorSet

Sets the outline color of a canvas widget.

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

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.

Description:
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.

5.2.3.25 CanvasOutlineOff

Disables outlining of a canvas widget.

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

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

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

Returns:
None.

5.2.3.26 CanvasOutlineOn

Enables outlining of a canvas widget.

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

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

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

Returns:
None.

5.2.3.27 CanvasStruct

Declares an initialized canvas widget data structure.

Definition:
```
#define CanvasStruct(pParent,
                     pNext,
                     pChild,
                     pDisplay,
                     lX,
                     lY,
                     lWidth,
                     lHeight,
                     ulStyle,
                     ulFillColor,
                     ulOutlineColor,
                     ulTextColor,
                     pFont,
                     pcText,
                     pucImage,
                     pfnOnPaint)
```

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.

Description:

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).
- **CANVAS_STYLE_TEXT_LEFT** to indicate that the canvas text should be left aligned within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_HCENTER** to indicate that the canvas text should be horizontally centered within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_RIGHT** to indicate that the canvas text should be right aligned within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_TOP** to indicate that the canvas text should be top aligned within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_VCENTER** to indicate that the canvas text should be vertically centered within the widget bounding rectangle.
- **CANVAS_STYLE_TEXT_BOTTOM** to indicate that the canvas text should be bottom aligned within the widget bounding rectangle.

Returns:

Nothing; this is not a function.

5.2.3.28 CanvasTextAlignment

Sets the text alignment for a canvas widget.

Definition:
```
#define CanvasTextAlignment(pWidget,
                            ulAlign)
```

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

    ulAlign contains the required text alignment setting.   This is a logical OR of style values **CANVAS_STYLE_TEXT_LEFT**, **CANVAS_STYLE_TEXT_RIGHT**, **CANVAS_STYLE_TEXT_HCENTER**, **CANVAS_STYLE_TEXT_VCENTER**, **CANVAS_STYLE_TEXT_TOP** and **CANVAS_STYLE_TEXT_BOTTOM**.

Description:
    This function sets the alignment of the text drawn inside the widget. Independent alignment options for horizontal and vertical placement allow the text to be positioned in one of 9 positions within the boinding box of the widget. The display is not updated until the next paint request.

Returns:
    None.

5.2.3.29 CanvasTextColorSet

Sets the text color of a canvas widget.

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

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.

Description:
    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.

5.2.3.30 CanvasTextOff

Disables the text on a canvas widget.

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

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

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

Returns:
> None.

5.2.3.31 CanvasTextOn

Enables the text on a canvas widget.

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

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

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

Returns:
> None.

5.2.3.32 CanvasTextOpaqueOff

Disables opaque text on a canvas widget.

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

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

Description:
> 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.

5.2.3.33 CanvasTextOpaqueOn

Enables opaque text on a canvas widget.

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

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

Description:
> 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.

5.2.3.34 CanvasTextSet

Changes the text drawn on a canvas widget.

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

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

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

Returns:
> None.

5.2.4   Function Documentation

5.2.4.1 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.

5.2.4.2   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, usBox-Size, 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   Detailed Description

The code for this widget is contained in `grlib/checkbox.c`, with `grlib/checkbox.h` containing the API definitions for use by applications.

6.2.2 Data Structure Documentation

6.2.2.1 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.

6.2.3 Define Documentation

6.2.3.1 CB_STYLE_FILL

Definition:
```
#define CB_STYLE_FILL
```

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

6.2.3.2   CB_STYLE_IMG

Definition:
```
#define CB_STYLE_IMG
```

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

6.2.3.3   CB_STYLE_OUTLINE

Definition:
```
#define CB_STYLE_OUTLINE
```

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

6.2.3.4   CB_STYLE_SELECTED

Definition:
```
#define CB_STYLE_SELECTED
```

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

6.2.3.5   CB_STYLE_TEXT

Definition:
```
#define CB_STYLE_TEXT
```

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

6.2.3.6   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).

6.2.3.7 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.

6.2.3.8 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.

6.2.3.9 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.

6.2.3.10 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.

6.2.3.11 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.

6.2.3.12 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.

6.2.3.13 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.

6.2.3.14 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.

6.2.3.15 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.

6.2.3.16 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.

6.2.3.17 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.

6.2.3.18 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.

6.2.3.19　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.

6.2.3.20　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.

6.2.3.21 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.

6.2.3.22 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.

6.2.3.23 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.

6.2.3.24 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.

6.2.3.25 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.

6.2.3.26 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.

6.2.4   Function Documentation

6.2.4.1   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.

6.2.4.2   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 Detailed Description

The code for this widget is contained in `grlib/container.c`, with `grlib/container.h` containing the API definitions for use by applications.

7.2.2 Data Structure Documentation

7.2.2.1 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.

7.2.3 Define Documentation

7.2.3.1 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.

7.2.3.2   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.

7.2.3.3   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.

7.2.3.4 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.

7.2.3.5 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.

7.2.3.6 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.

7.2.3.7 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.

7.2.3.8 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.

7.2.3.9 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 pointer 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.

7.2.3.10 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.

7.2.3.11 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.

7.2.3.12 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.

7.2.3.13 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.

7.2.3.14 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.

7.2.3.15 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.

7.2.3.16 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.

7.2.3.17 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.

7.2.3.18 CTR_STYLE_FILL

Definition:
```
#define CTR_STYLE_FILL
```

Description:
>   This flag indicates that the container widget should be filled.

7.2.3.19 CTR_STYLE_OUTLINE

Definition:
```
#define CTR_STYLE_OUTLINE
```

Description:
>   This flag indicates that the container widget should be outlined.

7.2.3.20 CTR_STYLE_TEXT

Definition:
```
#define CTR_STYLE_TEXT
```

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

7.2.3.21 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.

7.2.3.22 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).

7.2.4   Function Documentation

7.2.4.1   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.

7.2.4.2   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   Image Button Widget

8.1   Introduction

The image button widget provides a button that can be pressed, causing an action to be performed. An image button is defined using a background image, a pressed-state background image, a keycap image and, optionally, a text string. The use of independent background and keycap images can offer memory saving in some applications which wish to show many similar buttons.

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

- If style IB_STYLE_FILL is specified, the button area is filled with either the pressed or background color depending upon the button state.
- Unless style IB_STYLE_IMAGE_OFF is set, the pressed or unpressed state background image is drawn in the center of the button area.
- If a keycap image has been defined and style IB_STYLE_KEYCAP_OFF is not set, that image is drawn on top of the background image. If the button is in the released state, the keycap image is centered. If the button is pressed, the keycap image is offset by a number of pixels defined by the X and Y offset values currently specified for the widget.
- If style IB_STYLE_TEXT is set, the provided text string is drawn on top of the button. If the button is in the released state, the text is centered. If the button is pressed, the text is offset according to the X and Y offsets specified for the widget.

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

- tImageButtonWidget

Defines

- IB_STYLE_AUTO_REPEAT
- IB_STYLE_FILL

- IB_STYLE_IMAGE_OFF
- IB_STYLE_KEYCAP_OFF
- IB_STYLE_PRESSED
- IB_STYLE_RELEASE_NOTIFY
- IB_STYLE_TEXT
- ImageButton(sName, pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, ulStyle, ulForeColor, ulPressColor, ulBackColor, pFont, pcText, pucImage, pucPressImage, pucKeycapImage, sXOff, sYOff, usAutoRepeatDelay, usAutoRepeatRate, pfnOnClick)
- ImageButtonAutoRepeatDelaySet(pWidget, usDelay)
- ImageButtonAutoRepeatOff(pWidget)
- ImageButtonAutoRepeatOn(pWidget)
- ImageButtonAutoRepeatRateSet(pWidget, usRate)
- ImageButtonBackgroundColorSet(pWidget, ulColor)
- ImageButtonCallbackSet(pWidget, pfnOnClik)
- ImageButtonFillColorSet(pWidget, ulColor)
- ImageButtonFillOff(pWidget)
- ImageButtonFillOn(pWidget)
- ImageButtonForegroundColorSet(pWidget, ulColor)
- ImageButtonImageKeycapSet(pWidget, pImg)
- ImageButtonImageOff(pWidget)
- ImageButtonImageOn(pWidget)
- ImageButtonImagePressedSet(pWidget, pImg)
- ImageButtonImageSet(pWidget, pImg)
- ImageButtonKeycapOff(pWidget)
- ImageButtonKeycapOffsetSet(pWidget, sX, sY)
- ImageButtonKeycapOn(pWidget)
- ImageButtonPressedColorSet(pWidget, ulColor)
- ImageButtonStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, ulStyle, ulForeColor, ulPressColor, ulBackColor, pFont, pcText, pucImage, pucPressImage, pucKeycapImage, sXOff, sYOff, usAutoRepeatDelay, usAutoRepeatRate, pfnOnClick)
- ImageButtonTextOff(pWidget)
- ImageButtonTextOn(pWidget)
- ImageButtonTextSet(pWidget, pcTxt)

Functions

- void ImageButtonInit (tImageButtonWidget ∗pWidget, const tDisplay ∗pDisplay, long lX, long lY, long lWidth, long lHeight)
- long ImageButtonMsgProc (tWidget ∗pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)

8.2.1   Detailed Description

The code for this widget is contained in `grlib/imgbutton.c`, with `grlib/imgbutton.h` containing the API definitions for use by applications.

8.2.2   Data Structure Documentation

8.2.2.1   tImageButtonWidget

Definition:
```
typedef struct
{
    tWidget sBase;
    unsigned long ulStyle;
    unsigned long ulForegroundColor;
    unsigned long ulPressedColor;
    unsigned long ulBackgroundColor;
    const tFont *pFont;
    const char *pcText;
    const unsigned char *pucImage;
    const unsigned char *pucPressImage;
    const unsigned char *pucKeycapImage;
    short sXOffset;
    short sYOffset;
    unsigned short usAutoRepeatDelay;
    unsigned short usAutoRepeatRate;
    unsigned long ulAutoRepeatCount;
    void (*pfnOnClick)(tWidget *pWidget);
}
tImageButtonWidget
```

Members:

sBase The generic widget information.

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

ulForegroundColor The color to use for foreground pixels when a 1bpp image or text is in use. This value is ignored for all other image bit depths.

ulPressedColor The color to use for background pixels when the button is pressed and a 1bpp image is in use. This value is ignored for all other image bit depths. If IB_STYLE_FILL is specified, this is also the color that will be used to fill the widget when it is in the pressed state.

ulBackgroundColor The color to use for background pixels when the button is released and a 1bpp image is in use. This value is ignored for all other image bit depths. If IB_STYLE_FILL is specified, this is also the color that will be used to fill the widget when it is in the un-pressed state.

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

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

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

pucPressImage A pointer to the image to be drawn onto this image button when it is pressed.

pucKeycapImage A pointer to the image to be drawn above the background image for the button.

sXOffset The number of pixels to move the keycap image horizontally when the button is drawn in its pressed state.

sYOffset The number of pixels to move the keycap image vertically when the button is drawn in its pressed state.

usAutoRepeatDelay The number of pointer events to delay before starting to auto-repeat, if IB_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 IB_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 IB_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 IB_STYLE_AUTO_REPEAT is selected.

Description:

The structure that describes a image button widget.

8.2.3   Define Documentation

8.2.3.1   IB_STYLE_AUTO_REPEAT

Definition:
```
#define IB_STYLE_AUTO_REPEAT
```

Description:

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

8.2.3.2   IB_STYLE_FILL

Definition:
```
#define IB_STYLE_FILL
```

Description:

This flag indicates that the image button should be filled.

8.2.3.3   IB_STYLE_IMAGE_OFF

Definition:
```
#define IB_STYLE_IMAGE_OFF
```

Description:

This flag indicates that the background image is to be disabled.

8.2.3.4   IB_STYLE_KEYCAP_OFF

Definition:
```
#define IB_STYLE_KEYCAP_OFF
```

Description:

This flag indicates that the keycap image is to be disabled.

8.2.3.5   IB_STYLE_PRESSED

Definition:

```
#define IB_STYLE_PRESSED
```

Description:

This flag indicates that the image button is pressed.

8.2.3.6   IB_STYLE_RELEASE_NOTIFY

Definition:

```
#define IB_STYLE_RELEASE_NOTIFY
```

Description:

This flag indicates that the image button callback should be made when the button is released rather than when it is pressed. This does not affect the operation of auto repeat buttons.

8.2.3.7   IB_STYLE_TEXT

Definition:

```
#define IB_STYLE_TEXT
```

Description:

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

8.2.3.8   ImageButton

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

Definition:

```
#define ImageButton(sName,
                    pParent,
                    pNext,
                    pChild,
                    pDisplay,
                    lX,
                    lY,
                    lWidth,
                    lHeight,
                    ulStyle,
                    ulForeColor,
                    ulPressColor,
                    ulBackColor,
                    pFont,
```

```
                                pcText,
                                pucImage,
                                pucPressImage,
                                pucKeycapImage,
                                sXOff,
                                sYOff,
                                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 image button.

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

lWidth is the width of the image button.

lHeight is the height of the image button.

ulStyle is the style to be applied to the image button.

ulForeColor is the color to be used for foreground pixels when a 1bpp image is being drawn. It is ignored for all other image bit depths.

ulPressColor is the color to be used for foreground pixels when the button is pressed and a 1bpp image is being drawn. It is ignored for all other image bit depths.

ulBackColor is the color to be used for background pixels when the button is released and a 1bpp image is being drawn. It is ignored for all other image bit depths.

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

pcText is a pointer to the text to draw on this button.

pucImage is a pointer to the image to draw on the background of this image button when it is in the released state.

pucPressImage is a pointer to the image to draw on the background of this image button when it is in the pressed state.

pucKeycapImage is a pointer to the image to draw as the keycap of the on top of the image button, on top of the background image.

sXOff is the horizontal offset to apply when drawing the keycap image on the button when in the pressed state.

sYOff is the vertical offset to apply when drawing the keycap image on the button when in the pressed state.

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 image 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:

- **IB_STYLE_TEXT** to indicate that text should be drawn on the button.

- **IB_STYLE_FILL** to indicate that the background of the button should be filled with color.
- **IB_STYLE_KEYCAP_OFF** to indicate that the keycap image should not be drawn.
- **IB_STYLE_IMAGE_OFF** to indicate that the background image should not be drawn.
- **IB_STYLE_AUTO_REPEAT** to indicate that auto-repeat should be used.
- **IB_STYLE_RELEASE_NOTIFY** to indicate that the callback should be made when the button is released. If absent, the callback is called when the button is initially pressed.

Returns:
Nothing; this is not a function.

8.2.3.9   ImageButtonAutoRepeatDelaySet

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

Definition:
```
#define ImageButtonAutoRepeatDelaySet(pWidget,
                                      usDelay)
```

Parameters:
pWidget is a pointer to the image 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 image button is pressed.

Returns:
None.

8.2.3.10   ImageButtonAutoRepeatOff

Disables auto-repeat for a image button widget.

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

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

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

Returns:
None.

8.2.3.11   ImageButtonAutoRepeatOn

Enables auto-repeat for a image button widget.

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

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

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

Returns:
None.

8.2.3.12   ImageButtonAutoRepeatRateSet

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

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

Parameters:
pWidget is a pointer to the image 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 image button is pressed.

Returns:
None.

8.2.3.13   ImageButtonBackgroundColorSet

Sets the color of background pixels when using 1bpp images.

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

Parameters:
pWidget is a pointer to the image button widget to be modified.
ulColor is the background color to use.

Description:
This function changes the color that is used to draw background pixels when a 1bpp image is rendered on the button and the button is in the released state. The value is ignored for all other image bit depths. The display is not updated until the next paint request.

Returns:
    None.

8.2.3.14 ImageButtonCallbackSet

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

Definition:
```
#define ImageButtonCallbackSet(pWidget,
                               pfnOnClik)
```

Parameters:
    pWidget is a pointer to the image button widget to modify.
    pfnOnClik is a pointer to the function to call.

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

Returns:
    None.

8.2.3.15 ImageButtonFillColorSet

Sets the fill color of a image button widget.

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

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

Description:
    This function changes the color used to fill the background of the image button on the display. This is a duplicate of ImageButtonBackgroundColorSet which is left for backwards compatibility. The display is not updated until the next paint request.

Returns:
    None.

8.2.3.16 ImageButtonFillOff

Disables filling of a image button widget.

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

Parameters:
>**pWidget** is a pointer to the image button widget to modify.

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

Returns:
>None.

8.2.3.17   ImageButtonFillOn

Enables filling of a image button widget.

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

Parameters:
>**pWidget** is a pointer to the image button widget to modify.

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

Returns:
>None.

8.2.3.18   ImageButtonForegroundColorSet

Sets the color of foreground pixels when using 1bpp images.

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

Parameters:
>**pWidget** is a pointer to the image button widget to be modified.
>**ulColor** is the foreground color to use.

Description:
>This function changes the color that is used to draw foreground pixels when a 1bpp image or text string is rendered on the button. The value is ignored for all other image bit depths. The display is not updated until the next paint request.

Returns:
>None.

8.2.3.19 ImageButtonImageKeycapSet

Changes the keycap image drawn on a image button widget.

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

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

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

Returns:
None.

8.2.3.20 ImageButtonImageOff

Disables the background image for an image button widget.

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

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

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

Returns:
None.

8.2.3.21 ImageButtonImageOn

Enables the background image for an image button widget.

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

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

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

Returns:
None.

8.2.3.22 ImageButtonImagePressedSet

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

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

Parameters:
>*pWidget* is a pointer to the image button widget to be modified.
>*pImg* is a pointer to the image to draw onto the image button when it is pressed.

Description:
>This function changes the image that is drawn onto the background of the image button in its pressed state. The display is not updated until the next paint request.

Returns:
>None.

8.2.3.23 ImageButtonImageSet

Changes the image drawn on a image button widget.

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

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

Description:
>This function changes the image that is drawn onto the background of the image button in its unpressed state. The display is not updated until the next paint request.

Returns:
>None.

8.2.3.24 ImageButtonKeycapOff

Disables the keycap image for an image button widget.

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

Parameters:
>*pWidget* is a pointer to the image button widget to modify.

Description:
>This function disables the drawing of the keycap image on an image button widget. The display is not updated until the next paint request.

Returns:
> None.

8.2.3.25 ImageButtonKeycapOffsetSet

Changes the keycap image offset position on an image button widget.

Definition:
```
#define ImageButtonKeycapOffsetSet(pWidget,
                                   sX,
                                   sY)
```

Parameters:
> *pWidget* is a pointer to the image button widget to be modified.
>
> *sX* is the signed horizontal position offset for the keycap image when the image button is pressed. Positive values move the image right.
>
> *sY* is the signed vertical position offset for the keycap image when the image button is pressed. Positive values move the image down.

Description:
> This function changes the position that the keycap image is drawn at when the image button is pressed. The keycap image is moved iX pixels right and iY pixels down from the center position if the image button is pressed. This feature can be used to support 3D buttons. The display is not updated until the next paint request.

Returns:
> None.

8.2.3.26 ImageButtonKeycapOn

Enables the keycap image for an image button widget.

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

Parameters:
> *pWidget* is a pointer to the image button widget to modify.

Description:
> This function enables the drawing of the keycap image on an image button widget. The display is not updated until the next paint request.

Returns:
> None.

8.2.3.27 ImageButtonPressedColorSet

Sets the color of foreground pixels when the button is pressed and when using 1bpp images.

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

Parameters:

> *pWidget* is a pointer to the image button widget to be modified.
>
> *ulColor* is the pressed foreground color to use.

Description:

> This function changes the color that is used to draw foreground pixels when a 1bpp image is rendered on the button and the button is in the pressed state. The value is ignored for all other image bit depths. The display is not updated until the next paint request.

Returns:

> None.

8.2.3.28 ImageButtonStruct

Declares an initialized image button widget data structure.

Definition:
```
#define ImageButtonStruct(pParent,
                          pNext,
                          pChild,
                          pDisplay,
                          lX,
                          lY,
                          lWidth,
                          lHeight,
                          ulStyle,
                          ulForeColor,
                          ulPressColor,
                          ulBackColor,
                          pFont,
                          pcText,
                          pucImage,
                          pucPressImage,
                          pucKeycapImage,
                          sXOff,
                          sYOff,
                          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 image button.
>
> *lY* is the Y coordinate of the upper left corner of the image button.

lWidth is the width of the image button.

lHeight is the height of the image button.

ulStyle is the style to be applied to the image button.

ulForeColor is the color to be used for foreground pixels when a 1bpp image or text is being drawn. It is ignored for all other image bit depths.

ulPressColor is the color to be used for foreground pixels when the button is pressed and a 1bpp image is being drawn. It is ignored for all other image bit depths.

ulBackColor is the color to be used for background pixels when the button is released and a 1bpp image is being drawn. It is ignored for all other image bit depths.

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

pcText is a pointer to the text to draw on this button.

pucImage is a pointer to the image to draw on the background of this image button when it is in the released state.

pucPressImage is a pointer to the image to draw on the background of this image button when it is in the pressed state.

pucKeycapImage is a pointer to the image to draw as the keycap of the on top of the image button, on top of the background image.

sXOff is the horizontal offset to apply when drawing the keycap image on the button when in the pressed state.

sYOff is the vertical offset to apply when drawing the keycap image on the button when in the pressed state.

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 image 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:

```
tImageButtonWidget g_sImageButton = ImageButtonStruct(...);
```

Or, in an array of variables:

```
tImageButtonWidget g_psImageButtons[] =
{
    ImageButtonStruct(...),
    ImageButtonStruct(...)
};
```

ulStyle is the logical OR of the following:

- **IB_STYLE_TEXT** to indicate that text should be drawn on the button.
- **IB_STYLE_FILL** to indicate that the background of the button should be filled with color.
- **IB_STYLE_KEYCAP_OFF** to indicate that the keycap image should not be drawn.
- **IB_STYLE_IMAGE_OFF** to indicate that the background image should not be drawn.
- **IB_STYLE_AUTO_REPEAT** to indicate that auto-repeat should be used.
- **IB_STYLE_RELEASE_NOTIFY** to indicate that the callback should be made when the button is released. If absent, the callback is called when the button is initially pressed.

Returns:

Nothing; this is not a function.

8.2.3.29 ImageButtonTextOff

Disables the text on a image button widget.

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

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

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

Returns:
None.

8.2.3.30 ImageButtonTextOn

Enables the text on a image button widget.

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

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

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

Returns:
None.

8.2.3.31 ImageButtonTextSet

Changes the text drawn on a image button widget.

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

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

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

Returns:
None.

8.2.4   Function Documentation

8.2.4.1   ImageButtonInit

Initializes an image button widget.

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

Parameters:
>   ***pWidget*** is a pointer to the image 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 image button.
>   ***lY*** is the Y coordinate of the upper left corner of the image button.
>   ***lWidth*** is the width of the image button.
>   ***lHeight*** is the height of the image button.

Description:
>   This function initializes the provided image button widget.

Returns:
>   None.

8.2.4.2   ImageButtonMsgProc

Handles messages for an image button widget.

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

Parameters:
>   ***pWidget*** is a pointer to the image 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 image 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   ListBox Widget

9.1   Introduction

The listbox widget allows the user to select one from a list of several strings held by the widget. The touch screen can be used to select and deselect a string by tapping it or to scroll through the strings in the listbox by pressing and dragging on the screen. Whenever the selected element in the box changes, a message is sent to an application callback informing it of the new selection (or lack thereof).

A listbox may also be used as a passive indicator and, with minimal additional code, as a simple method of outputing scrolling text to the display.

When creating a listbox, the application provides an array of character pointers which will be used to hold the strings that the listbox displays. The application also provides the size of this array and indicates how many of its elements are already initialized. It is assumed that initialized entries always start at index 0 of the array.

Assuming an empty entry exists in the character pointer array, an application may add new entries to the listbox by using the function ListBoxTextAdd(). It may also replace any given string entry in the array by calling ListBoxTextSet() and providing the index of the entry to be replaced and a pointer to the new string.

When a listbox widget is drawn on the screen, the following sequence of operations occurs:

- The widget is outlined with its outline color if the **LISTBOX_STYLE_OUTLINE** flag is present in the widget style. If an outline is drawn, the area of the widget into which text will be drawn is reduced by 2 pixels on each side to ensure that the text does not interfere with the border.

- Strings are drawn into the visible portion of the widget starting at the top and continuing until either no more strings are available or the bottom of the widget is reached. The first string drawn depends upon whether the listbox content has been scrolled.

- Empty space to the right of each string and beneath the bottom of the last string drawn is filled with the widget background color.

When a pointer down message is received by the listbox, the widget checks to ensure that the pointer is within its boundary and, if so, remembers the Y coordinate of the press. When pointer move messages are received, the pointer Y coordinate is checked against the initial Y coordinate and, if more than 1 character height of movement is detected and the listbox contains more strings than can be displayed in the widget area, then the content of the box is scrolled upwards or downwards. When the pointer up message is received, if scrolling has occurred, the message is ignored. If no scrolling has taken place, however, the line of text beneath the pointer is selected or deselected (if it was initially selected) and a callback sent to the application informing it of the change.

Two additional style flags control the operation of the listbox widget. **LISTBOX_STYLE_LOCKED** causes the listbox not to make any application callbacks and to ignore user attempts to select or deselect entries. A locked list box does not, however, ignore all user input since it still responds

to pointer activity to allow the content to be scrolled. This style may be used in cases where, for example, the listbox is being used to report text status rather than as an interactive element.

LISTBOX_STYLE_WRAP is also typically used when the listbox is intended as a status reporting tool rather than as a method of offering a number of choices to the user. It indicates to the widget that the function ListBoxTextAdd() should discard the oldest string held by the widget if called when no more free entries exist in the widget's string table. Without this flag, an attempt to add more strings than the table can hold will result in an error being returned to the caller. **LISTBOX_STYLE_WRAP** allows a listbox widget to be used as a scrolling text display control. When the oldest string is discarded, the entry that will be drawn at the top of the listbox is incremented and this has the effect of scrolling the content by one line each time a new line of text is added.

9.2   Definitions

Data Structures

- tListBoxWidget

Defines

- ListBox(sName, pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, ulStyle, ulBgColor, ulSelBgColor, ulTextColor, ulSelTextColor, ulOutlineColor, pFont, ppcText, usMaxEntries, usPopulatedEntries, pfnOnChange)
- LISTBOX_STYLE_LOCKED
- LISTBOX_STYLE_OUTLINE
- LISTBOX_STYLE_WRAP
- ListBoxBackgroundColorSet(pWidget, ulColor)
- ListBoxCallbackSet(pWidget, pfnCallback)
- ListBoxClear(pWidget)
- ListBoxFontSet(pWidget, pFnt)
- ListBoxLock(pWidget)
- ListBoxOutlineColorSet(pWidget, ulColor)
- ListBoxOutlineOff(pWidget)
- ListBoxOutlineOn(pWidget)
- ListBoxSelectedBackgroundColorSet(pWidget, ulColor)
- ListBoxSelectedTextColorSet(pWidget, ulColor)
- ListBoxSelectionGet(pWidget)
- ListBoxSelectionSet(pWidget, sSel)
- ListBoxStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, ulStyle, ulBgColor, ulSelBgColor, ulTextColor, ulSelTextColor, ulOutlineColor, pFont, ppcText, usMaxEntries, usPopulatedEntries, pfnOnChange)
- ListBoxTextColorSet(pWidget, ulColor)
- ListBoxTextSet(pWidget, pcTxt, ulIndex)
- ListBoxUnlock(pWidget)
- ListBoxWrapDisable(pWidget)
- ListBoxWrapEnable(pWidget)

Functions

- void ListBoxInit (tListBoxWidget ∗pWidget, const tDisplay ∗pDisplay, const char ∗∗ppcText, unsigned short usMaxEntries, unsigned short usPopulatedEntries, long lX, long lY, long lWidth, long lHeight)
- long ListBoxMsgProc (tWidget ∗pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)
- long ListBoxTextAdd (tListBoxWidget ∗pListBox, const char ∗pcTxt)

9.2.1 Detailed Description

The code for this widget is contained in `grlib/listbox.c`, with `grlib/listbox.h` containing the API definitions for use by applications.

9.2.2 Data Structure Documentation

9.2.2.1 tListBoxWidget

Definition:
```
typedef struct
{
    tWidget sBase;
    unsigned long ulStyle;
    unsigned long ulBackgroundColor;
    unsigned long ulSelectedBackgroundColor;
    unsigned long ulTextColor;
    unsigned long ulSelectedTextColor;
    unsigned long ulOutlineColor;
    const tFont *pFont;
    const char **ppcText;
    unsigned short usMaxEntries;
    unsigned short usPopulated;
    short sSelected;
    unsigned short usStartEntry;
    unsigned short usOldestEntry;
    unsigned short usScrolled;
    long lPointerY;
    void (*pfnOnChange)(tWidget *pWidget,
                        short sSelIndex);
}
tListBoxWidget
```

Members:
> ***sBase*** The generic widget information.
>
> ***ulStyle*** The style for this widget. This is a set of flags defined by LISTBOX_STYLE_xxx.
>
> ***ulBackgroundColor*** The 24-bit RGB color used as the background for the listbox.
>
> ***ulSelectedBackgroundColor*** The 24-bit RGB color used as the background for the selected entry in the listbox.
>
> ***ulTextColor*** The 24-bit RGB color used to draw text on this listbox.

ulSelectedTextColor The 24-bit RGB color used to draw the selected text on this listbox.

ulOutlineColor The 24-bit RGB color used to outline this listbox, if LIST-BOX_STYLE_OUTLINE is selected.

pFont A pointer to the font used to render the listbox text.

ppcText A pointer to the array of string pointers representing the contents of the list box.

usMaxEntries The number of elements in the array pointed to by pccText.

usPopulated The number of elements in the array pointed to by pccText which are currently populated with strings.

sSelected The index of the string currently selected in the list box. If no selection has been made, this will be set to 0xFFFF (-1).

usStartEntry The index of the string that appears at the top of the list box. This is used by the widget class to control scrolling of the box content. This is an internal variable and must not be modified by an application using this widget class.

usOldestEntry The index of the oldest entry in the ppcText array. This is used by the widget class to determine where to add a new string if the array is full and the listbox has style LISTBOX_STYLE_WRAP. This is an internal variable and must not be modified by an application using this widget class.

usScrolled A flag which we use to determine whether to change the selected element when the pointer is lifted. The listbox will change the selection if no scrolling was performed since the last WIDGET_MSG_PTR_DOWN was received. This is an internal variable and must not be modified by an application using this widget class.

lPointerY The Y coordinate of the last pointer position we received. This is an internal variable used to manage scrolling of the listbox contents and must not be modified by an application using this widget class.

pfnOnChange A pointer to the application-supplied callback function. This function will be called each time the selected element in the list box changes. The sSelIndex parameter contains the index of the selected string in ppcText array or, if no element is selected, 0xFFFF (-1).

Description:

The structure that describes a listbox widget.

9.2.3   Define Documentation

9.2.3.1   ListBox

Declares an initialized variable containing a listbox widget data structure.

Definition:

```
#define ListBox(sName,
                pParent,
                pNext,
                pChild,
                pDisplay,
                lX,
                lY,
                lWidth,
                lHeight,
                ulStyle,
```

```
                    ulBgColor,
                    ulSelBgColor,
                    ulTextColor,
                    ulSelTextColor,
                    ulOutlineColor,
                    pFont,
                    ppcText,
                    usMaxEntries,
                    usPopulatedEntries,
                    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 listbox.

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

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

lWidth is the width of the listbox.

lHeight is the height of the listbox.

ulStyle is the style to be applied to the listbox.

ulBgColor is the background color for the listbox.

ulSelBgColor is the background color for the selected element in the listbox.

ulTextColor is the color used to draw text on the listbox.

ulSelTextColor is the color used to draw the selected element text in the listbox.

ulOutlineColor is the color used to outline the listbox.

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

ppcText is a pointer to the string table for the listbox.

usMaxEntries provides the number of entries in the *ppcText* array and represents the maximum number of strings the listbox can hold.

usPopulatedEntries indicates the number of entries in the *ppcText* array that currently hold valid string for the listbox.

pfnOnChange is a pointer to the application callback for the listbox.

Description:

This macro declares a variable containing an initialized listbox 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:

- **LISTBOX_STYLE_OUTLINE** to indicate that the listbox should be outlined.
- **LISTBOX_STYLE_LOCKED** to indicate that the listbox should ignore user input and merely display its contents.
- **LISTBOX_STYLE_WRAP** to indicate that the listbox should discard the oldest string it contains if asked to add a new string while the string table is already full.

Returns:

Nothing; this is not a function.

9.2.3.2   LISTBOX_STYLE_LOCKED

Definition:
```
#define LISTBOX_STYLE_LOCKED
```

Description:
This flag indicates that the listbox is not interactive but merely displays strings. Scrolling of the listbox content is supported when this flag is set but widgets using this style do not make callbacks to the application and do not support selection and deselection of entries. This may be used if a listbox is intended, for example, as a text output or status reporting control.

9.2.3.3   LISTBOX_STYLE_OUTLINE

Definition:
```
#define LISTBOX_STYLE_OUTLINE
```

Description:
This flag indicates that the listbox should be outlined. If enabled, the widget is drawn with a two pixel border, the outer, single pixel rectangle of which is in the color found in the ulOutlineColor field of the widget structure and the inner rectangle in color ulBackgroundColor.

9.2.3.4   LISTBOX_STYLE_WRAP

Definition:
```
#define LISTBOX_STYLE_WRAP
```

Description:
This flag controls the behavior of the listbox if a new string is added when the string table (ppcText) is already full. If this style is set, the oldest string in the table is replaced with new one and, if the discarded string was currently displayed, the display positions will be fixed up to ensure that the (new) oldest string remains at the top of the listbox. If this style is not set, the attempt to set a new string will fail if the table is full.

9.2.3.5   ListBoxBackgroundColorSet

Sets the background color of a listbox widget.

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

Parameters:
pWidget is a pointer to the listbox widget to be modified.
ulColor is the 24-bit RGB color to use for the listbox background.

Description:
This function changes the color used for the listbox background on the display. The display is not updated until the next paint request.

Returns:
None.

9.2.3.6 ListBoxCallbackSet

Sets the function to call when the listbox selection changes.

Definition:
```
#define ListBoxCallbackSet(pWidget,
                           pfnCallback)
```

Parameters:
pWidget is a pointer to the listbox widget to modify.
pfnCallback is a pointer to the function to call.

Description:
This function sets the function to be called when the selected element in this listbox changes. If style **LISTBOX_STYLE_LOCKED** is selected, or the callback function pointer set is NULL, no callbacks will be made.

Returns:
None.

9.2.3.7 ListBoxClear

Empties the listbox.

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

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

Description:
This function removes all text from a listbox widget. The display is not updated until the next paint request.

Returns:
None.

9.2.3.8 ListBoxFontSet

Sets the font for a listbox widget.

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

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

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

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

Returns:
None.

9.2.3.9   ListBoxLock

Locks a listbox making it ignore attempts to select elements.

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

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

Description:
This function locks a listbox widget and makes it ignore attempts to select or deselect an element. When locked, a listbox acts as a passive indicator. Strings may be added and the selected element changed via calls to ListBoxSelectioSet() but pointer activity will not change the selection and no callbacks will be made. In this mode, the user may still use the pointer to scroll the content of the listbox assuming it contains more strings that can be displayed in the widget area.

Returns:
None.

9.2.3.10   ListBoxOutlineColorSet

Sets the outline color of a listbox widget.

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

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

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

Returns:
None.

9.2.3.11 ListBoxOutlineOff

Disables outlining of a listbox widget.

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

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

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

Returns:
None.

9.2.3.12 ListBoxOutlineOn

Enables outlining of a listbox widget.

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

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

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

Returns:
None.

9.2.3.13 ListBoxSelectedBackgroundColorSet

Sets the background color of the selected element in a listbox widget.

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

Parameters:
pWidget is a pointer to the listbox widget to be modified.
ulColor is the 24-bit RGB color to use for the background of the selected element.

Description:
This function changes the color used for the background of the selected line of text on the display. The display is not updated until the next paint request.

Returns:
None.

9.2.3.14 ListBoxSelectedTextColorSet

Sets the text color of the selected element in a listbox widget.

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

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

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

Returns:
> None.

9.2.3.15 ListBoxSelectionGet

Gets the index of the current selection within the listbox.

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

Parameters:
> ***pWidget*** is a pointer to the listbox widget to be queried.

Description:
> This function returns the index of the item currently selected in a listbox. If no selection has been made, 0xFFFF (-1) is returned.

Returns:
> None.

9.2.3.16 ListBoxSelectionSet

Sets the current selection within the listbox.

Definition:
```
#define ListBoxSelectionSet(pWidget,
                            sSel)
```

Parameters:
> ***pWidget*** is a pointer to the listbox widget to modify.
> ***sSel*** is the index of the item to select.

Description:
> This function selects an item within the list box. The display is not updated until the next paint request.

Returns:
    None.

9.2.3.17 ListBoxStruct

Declares an initialized listbox widget data structure.

Definition:
```
#define ListBoxStruct(pParent,
                      pNext,
                      pChild,
                      pDisplay,
                      lX,
                      lY,
                      lWidth,
                      lHeight,
                      ulStyle,
                      ulBgColor,
                      ulSelBgColor,
                      ulTextColor,
                      ulSelTextColor,
                      ulOutlineColor,
                      pFont,
                      ppcText,
                      usMaxEntries,
                      usPopulatedEntries,
                      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 listbox.
    lX is the X coordinate of the upper left corner of the listbox.
    lY is the Y coordinate of the upper left corner of the listbox.
    lWidth is the width of the listbox.
    lHeight is the height of the listbox.
    ulStyle is the style to be applied to the listbox.
    ulBgColor is the background color for the listbox.
    ulSelBgColor is the background color for the selected element in the listbox.
    ulTextColor is the color used to draw text on the listbox.
    ulSelTextColor is the color used to draw the selected element text in the listbox.
    ulOutlineColor is the color used to outline the listbox.
    pFont is a pointer to the font to be used to draw text on the listbox.
    ppcText is a pointer to the string table for the listbox.
    usMaxEntries provides the number of entries in the *ppcText* array and represents the maximum number of strings the listbox can hold.
    usPopulatedEntries indicates the number of entries in the *ppcText* array that currently hold valid string for the listbox.

pfnOnChange is a pointer to the application callback for the listbox.

Description:

This macro provides an initialized listbox 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:

```
tListBoxWidget g_sListBox = ListBoxStruct(...);
```

Or, in an array of variables:

```
tListBoxWidget g_psListBox[] =
{
    ListBoxStruct(...),
    ListBoxStruct(...)
};
```

ulStyle is the logical OR of the following:

- **LISTBOX_STYLE_OUTLINE** to indicate that the listbox should be outlined.
- **LISTBOX_STYLE_LOCKED** to indicate that the listbox should ignore user input and merely display its contents.
- **LISTBOX_STYLE_WRAP** to indicate that the listbox should discard the oldest string it contains if asked to add a new string while the string table is already full.

Returns:

Nothing; this is not a function.

9.2.3.18 ListBoxTextColorSet

Sets the text color of a listbox widget.

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

Parameters:

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

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

Description:

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

Returns:

None.

9.2.3.19 ListBoxTextSet

Changes the text associated with an element in the listbox widget.

Definition:
```
#define ListBoxTextSet(pWidget,
                       pcTxt,
                       ulIndex)
```

Parameters:
>   ***pWidget*** is a pointer to the listbox widget to be modified.
>   ***pcTxt*** is a pointer to the new text string.
>   ***ulIndex*** is the index of the element whose string is to be replaced.

Description:
>   This function replaces the string associated with one of the listbox elements. This call should only be used to replace a string for an already-populated element. To add a new string, use ListBoxTextAdd(). The display is not updated until the next paint request.

Returns:
>   None.

9.2.3.20 ListBoxUnlock

Unlocks a listbox making it respond to pointer input.

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

Parameters:
>   ***pWidget*** is a pointer to the listbox widget to modify.

Description:
>   This function unlocks a listbox widget. When unlocked, a listbox will respond to pointer input by setting its selected element appropriately and informing the application of changes via callbacks.

Returns:
>   None.

9.2.3.21 ListBoxWrapDisable

Disables text wrapping in a listbox.

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

Parameters:
>   ***pWidget*** is a pointer to the listbox widget to modify.

Description:
>   This function disables text wrapping in a listbox widget. With wrapping enabled, calls to ListBoxTextAdd() made when the widget string table is full will discard the oldest string in favor of the new one. If wrapping is disabled, these calls will fail.

Returns:
>   None.

9.2.3.22 ListBoxWrapEnable

Enables wrapping in a listbox.

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

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

Description:
This function enables text wrapping in a listbox widget. With wrapping enabled, calls to List-
BoxTextAdd() made when the widget string table is full will discard the oldest string in favor of
the new one. If wrapping is disabled, these calls will fail.

Returns:
None.

9.2.4   Function Documentation

9.2.4.1   ListBoxInit

Initializes a listbox widget.

Prototype:
```
void
ListBoxInit(tListBoxWidget *pWidget,
            const tDisplay *pDisplay,
            const char **ppcText,
            unsigned short usMaxEntries,
            unsigned short usPopulatedEntries,
            long lX,
            long lY,
            long lWidth,
            long lHeight)
```

Parameters:
pWidget is a pointer to the listbox widget to initialize.

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

ppcText is a pointer to an array of character pointers which will hold the strings that the listbox
displays.

usMaxEntries provides the total number of entries in the *ppcText* array.

usPopulatedEntries provides the number of entries in the *ppcText* array which are populated.

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

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

lWidth is the width of the listbox.

lHeight is the height of the listbox.

Description:
This function initializes the provided listbox widget.

Returns:
> None.

9.2.4.2   ListBoxMsgProc

Handles messages for a listbox widget.

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

Parameters:
> **pWidget** is a pointer to the listbox 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 listbox 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.

9.2.4.3   ListBoxTextAdd

Adds a line of text to a listbox.

Prototype:
```
long
ListBoxTextAdd(tListBoxWidget *pListBox,
               const char *pcTxt)
```

Parameters:
> **pListBox** is a pointer to the listbox widget that is to receive the new text string.
>
> **pcTxt** is a pointer to the string that is to be added to the listbox.

Description:
> This function adds a new string to the listbox. If the listbox has style **LISTBOX_STYLE_WRAP** and the current string table is full, this function will discard the oldest string and replace it with the one passed here. If this style flag is absent, the function will return -1 if no empty entries exist in the string table for the widget.
>
> The display is not automatically updated as a result of this function call. An application must call WidgetPaint() to update the display after adding a new string to the listbox.

Note:

To replace the string associated with a particular, existing element in the listbox, use ListBox-TextSet().

Returns:

Returns the string table index into which the new string has been placed if successful or -1 if the string table is full and **LISTBOX_STYLE_WRAP** is not set.

10 Push Button Widget

10.1 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 with 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 circle. 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).

10.2 Definitions

Data Structures

- tPushButtonWidget

Defines

- **CircularButton**(sName, pParent, pNext, pChild, pDisplay, lX, lY, lR, ulStyle, ulFillColor, ulPress-FillColor, ulOutlineColor, ulTextColor, pFont, pcText, pucImage, pucPressImage, usAutoRepeatDelay, usAutoRepeatRate, pfnOnClick)
- **CircularButtonStruct**(pParent, pNext, pChild, pDisplay, lX, lY, lR, ulStyle, ulFillColor, ulPress-FillColor, 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_RELEASE_NOTIFY**
- **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 ul-Param1, 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 ul-Param1, unsigned long ulParam2)

10.2.1 Detailed Description

The code for this widget is contained in `grlib/pushbutton.c`, with `grlib/pushbutton.h` containing the API definitions for use by applications.

10.2.2 Data Structure Documentation

10.2.2.1 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.

10.2.3 Define Documentation

10.2.3.1 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.
- **PB_STYLE_RELEASE_NOTIFY** to indicate that the callback should be made when the button is released. If absent, the callback is called when the button is initially pressed.

Returns:

Nothing; this is not a function.

10.2.3.2   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.
- **PB_STYLE_RELEASE_NOTIFY** to indicate that the callback should be made when the button is released. If absent, the callback is called when the button is initially pressed.

Returns:
Nothing; this is not a function.

10.2.3.3 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.

10.2.3.4 PB_STYLE_FILL

Definition:
```
#define PB_STYLE_FILL
```

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

10.2.3.5 PB_STYLE_IMG

Definition:
```
#define PB_STYLE_IMG
```

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

10.2.3.6 PB_STYLE_OUTLINE

Definition:
```
#define PB_STYLE_OUTLINE
```

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

10.2.3.7 PB_STYLE_PRESSED

Definition:
```
#define PB_STYLE_PRESSED
```

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

10.2.3.8 PB_STYLE_RELEASE_NOTIFY

Definition:
```
#define PB_STYLE_RELEASE_NOTIFY
```

Description:
This flag indicates that the push button callback should be made when the button is released rather than when it is pressed. This does not affect the operation of auto repeat buttons.

10.2.3.9 PB_STYLE_TEXT

Definition:
```
#define PB_STYLE_TEXT
```

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

10.2.3.10 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).

10.2.3.11 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.

10.2.3.12 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.

10.2.3.13 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.

10.2.3.14 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.

10.2.3.15 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.

10.2.3.16 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.

10.2.3.17 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.

10.2.3.18 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.

10.2.3.19 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.

10.2.3.20 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.

10.2.3.21 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.

10.2.3.22 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.

10.2.3.23 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.

10.2.3.24 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.

10.2.3.25 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.

10.2.3.26 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.

10.2.3.27 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.

10.2.3.28 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.

10.2.3.29 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.

10.2.3.30 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.

10.2.3.31 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.

10.2.3.32 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.

10.2.3.33 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.

10.2.3.34 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.

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

IY 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.
- **PB_STYLE_RELEASE_NOTIFY** to indicate that the callback should be made when the button is released. If absent, the callback is called when the button is initially pressed.

Returns:

Nothing; this is not a function.

10.2.3.35 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.
- **PB_STYLE_RELEASE_NOTIFY** to indicate that the callback should be made when the button is released. If absent, the callback is called when the button is initially pressed.

Returns:
Nothing; this is not a function.

10.2.4 Function Documentation

10.2.4.1 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.

10.2.4.2 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.

10.2.4.3 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.

10.2.4.4 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.

11   Radio Button Widget

11.1   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) is a convenient widget to use as a parent when defining a group of radio buttons. Using the various styles it supports, the container may be used, for example, to draw a border around the button group or place the group on a colored background.

11.2   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)

11.2.1 Detailed Description

The code for this widget is contained in `grlib/radiobutton.c`, with `grlib/radiobutton.h` containing the API definitions for use by applications.

11.2.2 Data Structure Documentation

11.2.2.1 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.

11.2.3   Define Documentation

11.2.3.1   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.

11.2.3.2 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.

11.2.3.3 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.

11.2.3.4 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.

11.2.3.5 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.

11.2.3.6 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.

11.2.3.7 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.

11.2.3.8 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.

11.2.3.9 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.

11.2.3.10 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.

11.2.3.11 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.

11.2.3.12 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.

11.2.3.13 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.

11.2.3.14 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.

11.2.3.15 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.

11.2.3.16 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.

11.2.3.17 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.

11.2.3.18 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.

11.2.3.19 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.

11.2.3.20 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.

11.2.3.21 RB_STYLE_FILL

Definition:

```
#define RB_STYLE_FILL
```

Description:

This flag indicates that the radio button should be filled.

11.2.3.22 RB_STYLE_IMG

Definition:

```
#define RB_STYLE_IMG
```

Description:

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

11.2.3.23 RB_STYLE_OUTLINE

Definition:

```
#define RB_STYLE_OUTLINE
```

Description:

This flag indicates that the radio button should be outlined.

11.2.3.24 RB_STYLE_SELECTED

Definition:

```
#define RB_STYLE_SELECTED
```

Description:

This flag indicates that the radio button is selected.

11.2.3.25 RB_STYLE_TEXT

Definition:

```
#define RB_STYLE_TEXT
```

Description:

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

11.2.3.26 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).

11.2.4   Function Documentation

11.2.4.1   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.

11.2.4.2   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.

12   Slider Widget

12.1   Introduction

The slider widget allows the user to drag a marker either horizontally or vertically to select a value from within an application-supplied range.

A slider consists of two distinct areas - an active or foreground area representing the current value of the control and a background area occupying the remainder of the widget rectangle. Each of these areas may be filled with a color, have an image drawn in them and have text rendered. A separate image may be drawn in each of the two areas (though each image is centered on the widget as a whole so that the images are revealed or obscured depending upon the slider position) and different fill and text colors may also be used. The widget may contain a single text string which is centered within the widget rectangle but the visibility and color of the text in each portion of the widget is selectable by the application. Additionally, the widget may be outlined in a color chosen by the application.

The range of values represented by the slider is independent of the displayed size of the slider widget. The application determines the lower and upper values that the slider should report and the widget translates the position of the slider on the display into a value within that range. Obviously, the granularity of values that the slider reports will vary with the size of the actual widget, however. For example, a horizontal slider displayed using a 100 pixel wide widget can have its range set to [0,99] resulting in a granularity of 1 (since there are 100 pixels on the display to represent the 100 integers in the range). The same widget with its range set to [-100, 99] would have a granularity of 2 since the range contains twice as many possible values as there are screen pixel positions to represent them.

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

- The slider is outlined with its outline color if the **SL_STYLE_OUTLINE** flag is present in the widget style.

- The current slider value is converted into a position which defines the split between the active (foreground) and background areas of the widget. Each of these areas are then drawn separately.

- The slider active region is filled with a color if the **SL_STYLE_FILL** flag is present. The color used is that provided in the *ulFillColor* parameter to the macro used to create the widget.

- The slider foreground image is drawn centered within the widget and clipped to the active rectangle. The **SL_STYLE_IMG** flag enables image rendering in the active area.

- The slider text is drawn centered within the widget and clipped to the active rectangle. The text is drawn in the color provided in the *ulTextColor* parameter to the macro used to create the widget. The **SL_STYLE_TEXT** flag enables text rendering in the active area and **SL_STYLE_TEXT_OPAQUE** controls whether the background to the text is opaque or transparent.

■ The slider background region is filled with a color if the **SL_STYLE_BACKG_FILL** flag is present. The color used is that provided in the *ulBackgroundFillColor* parameter to the macro used to create the widget.

■ The slider background image is drawn centered within the widget and clipped to the background rectangle. The **SL_STYLE_BACKG_IMG** flag enables image rendering in the background area.

■ The slider text is drawn centered within the widget and clipped to the background rectangle. The text is drawn in the color provided in the *ulBackgroundTextColor* parameter to the macro used to create the widget. The **SL_STYLE_BACKG_TEXT** flag enables text rendering in the background area and **SL_STYLE_BACKG_TEXT_OPAQUE** controls whether the background to the text is opaque or transparent.

These steps are cumulative and any combination of these styles can be selected simultaneously. So, for example, the slider can be outlined, filled, have separate images drawn in each of the areas, and have text rendered on top.

When a pointer down message is received by the slider, the widget checks to ensure that the pointer is within its boundary and, if so, translates the pointer position into a slider value based on the minimum and maximum values that the slider is set to represent. This value is then used to redraw the slider at the relevant position and the application callback, if present, is called to provide information on the new value. When a pointer move message is received, the same processing occurs with the exception that the boundary check is not performed. This allows the slider to be moved to the full extent of its range by dragging past the displayed ends of the widget. The widget will ensure that the values reported always fall within the desired range.

12.2 Definitions

Data Structures

■ tSliderWidget

Defines

■ SL_STYLE_BACKG_FILL
■ SL_STYLE_BACKG_IMG
■ SL_STYLE_BACKG_TEXT
■ SL_STYLE_BACKG_TEXT_OPAQUE
■ SL_STYLE_FILL
■ SL_STYLE_IMG
■ SL_STYLE_LOCKED
■ SL_STYLE_OUTLINE
■ SL_STYLE_TEXT
■ SL_STYLE_TEXT_OPAQUE
■ SL_STYLE_VERTICAL

- Slider(sName, pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, lMin, lMax, lValue, ul-Style, ulFillColor, ulBackgroundFillColor, ulOutlineColor, ulTextColor, ulBackgroundTextColor, pFont, pcText, pucImage, pucBackgroundImage, pfnOnChange)
- SliderBackgroundFillOff(pWidget)
- SliderBackgroundFillOn(pWidget)
- SliderBackgroundImageOff(pWidget)
- SliderBackgroundImageOn(pWidget)
- SliderBackgroundImageSet(pWidget, pImg)
- SliderBackgroundTextColorSet(pWidget, ulColor)
- SliderBackgroundTextOff(pWidget)
- SliderBackgroundTextOn(pWidget)
- SliderBackgroundTextOpaqueOff(pWidget)
- SliderBackgroundTextOpaqueOn(pWidget)
- SliderCallbackSet(pWidget, pfnCallback)
- SliderFillColorBackgroundedSet(pWidget, ulColor)
- SliderFillColorSet(pWidget, ulColor)
- SliderFillOff(pWidget)
- SliderFillOn(pWidget)
- SliderFontSet(pWidget, pFnt)
- SliderImageOff(pWidget)
- SliderImageOn(pWidget)
- SliderImageSet(pWidget, pImg)
- SliderLock(pWidget)
- SliderOutlineColorSet(pWidget, ulColor)
- SliderOutlineOff(pWidget)
- SliderOutlineOn(pWidget)
- SliderRangeSet(pWidget, lMinimum, lMaximum)
- SliderStruct(pParent, pNext, pChild, pDisplay, lX, lY, lWidth, lHeight, lMin, lMax, lValue, ulStyle, ulFillColor, ulBackgroundFillColor, ulOutlineColor, ulTextColor, ulBackgroundTextColor, pFont, pcText, pucImage, pucBackgroundImage, pfnOnChange)
- SliderTextColorSet(pWidget, ulColor)
- SliderTextOff(pWidget)
- SliderTextOn(pWidget)
- SliderTextOpaqueOff(pWidget)
- SliderTextOpaqueOn(pWidget)
- SliderTextSet(pWidget, pcTxt)
- SliderUnlock(pWidget)
- SliderValueSet(pWidget, lVal)
- SliderVerticalSet(pWidget, bVertical)

Functions

- void SliderInit (tSliderWidget *pWidget, const tDisplay *pDisplay, long lX, long lY, long lWidth, long lHeight)
- long SliderMsgProc (tWidget *pWidget, unsigned long ulMsg, unsigned long ulParam1, unsigned long ulParam2)

12.2.1   Detailed Description

The code for this widget is contained in `grlib/slider.c`, with `grlib/slider.h` containing the API definitions for use by applications.

12.2.2   Data Structure Documentation

12.2.2.1   tSliderWidget

Definition:
```
typedef struct
{
    tWidget sBase;
    unsigned long ulStyle;
    unsigned long ulFillColor;
    unsigned long ulBackgroundFillColor;
    unsigned long ulOutlineColor;
    unsigned long ulTextColor;
    unsigned long ulBackgroundTextColor;
    const tFont *pFont;
    const char *pcText;
    const unsigned char *pucImage;
    const unsigned char *pucBackgroundImage;
    void (*pfnOnChange)(tWidget *pWidget,
                        long lValue);
    long lMin;
    long lMax;
    long lValue;
    short sPos;
}
tSliderWidget
```

Members:

sBase The generic widget information.

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

ulFillColor The 24-bit RGB color used to fill this slider, if SL_STYLE_FILL is selected, and to use as the background color if SL_STYLE_TEXT_OPAQUE is selected.

ulBackgroundFillColor The 24-bit RGB color used to fill the background portion of the slider if SL_STYLE_FILL is selected, and to use as the background color if SL_STYLE_TEXT_OPAQUE is selected.

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

ulTextColor The 24-bit RGB color used to draw text on the "active" portion of this slider, if SL_STYLE_TEXT is selected.

ulBackgroundTextColor The 24-bit RGB color used to draw text on the background portion of this slider, if SL_STYLE_TEXT is selected.

pFont A pointer to the font used to render the slider text, if SL_STYLE_TEXT is selected.

pcText A pointer to the text to draw on this slider, if SL_STYLE_TEXT is selected.

pucImage A pointer to the image to be drawn onto this slider, if SL_STYLE_IMG is selected.

pucBackgroundImage A pointer to the image to be drawn onto this slider background if SL_STYLE_BACKG_IMG is selected.

pfnOnChange A pointer to the function to be called when the state of the slider changes.

lMin The value represented by the slider at its zero position. This value is returned if a horizontal slider is pulled to the far left or a vertical slider is pulled to the bottom of widget's bounding rectangle.

lMax The value represented by the slider at its maximum position. This value is returned if a horizontal slider is pulled to the far right or a vertical slider is pulled to the top of the widget's bounding rectangle.

lValue The current slider value scaled according to the minimum and maximum values for the control.

sPos This internal work variable stores the pixel position representing the current slider value.

Description:
The structure that describes a slider widget.

12.2.3   Define Documentation

12.2.3.1 SL_STYLE_BACKG_FILL

Definition:
```
#define SL_STYLE_BACKG_FILL
```

Description:
This flag indicates that the background portion of the slider should be filled.

12.2.3.2 SL_STYLE_BACKG_IMG

Definition:
```
#define SL_STYLE_BACKG_IMG
```

Description:
This flag indicates that the slider should have an image drawn on its background.

12.2.3.3 SL_STYLE_BACKG_TEXT

Definition:
```
#define SL_STYLE_BACKG_TEXT
```

Description:
This flag indicates that the slider should have text drawn on top of the background portion.

12.2.3.4 SL_STYLE_BACKG_TEXT_OPAQUE

Definition:
```
#define SL_STYLE_BACKG_TEXT_OPAQUE
```

Description:

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

12.2.3.5 SL_STYLE_FILL

Definition:

```
#define SL_STYLE_FILL
```

Description:

This flag indicates that the active portion of the slider should be filled.

12.2.3.6 SL_STYLE_IMG

Definition:

```
#define SL_STYLE_IMG
```

Description:

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

12.2.3.7 SL_STYLE_LOCKED

Definition:

```
#define SL_STYLE_LOCKED
```

Description:

This flag causes the slider to ignore pointer input and act as a passive indicator. An application may set its value and repaint it as normal but its value will not be changed in response to any touchscreen activity.

12.2.3.8 SL_STYLE_OUTLINE

Definition:

```
#define SL_STYLE_OUTLINE
```

Description:

This flag indicates that the slider should be outlined.

12.2.3.9 SL_STYLE_TEXT

Definition:

```
#define SL_STYLE_TEXT
```

Description:

This flag indicates that the slider should have text drawn on top of the active portion.

12.2.3.10 SL_STYLE_TEXT_OPAQUE

Definition:
```
#define SL_STYLE_TEXT_OPAQUE
```

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

12.2.3.11 SL_STYLE_VERTICAL

Definition:
```
#define SL_STYLE_VERTICAL
```

Description:
This flag indicates that the slider is vertical rather than horizontal. If the flag is absent, the slider is assumed to operate horizontally with the reported value increasing from left to right. If set, the reported value increases from the bottom of the widget towards the top.

12.2.3.12 Slider

Declares an initialized variable containing a slider widget data structure.

Definition:
```
#define Slider(sName,
               pParent,
               pNext,
               pChild,
               pDisplay,
               lX,
               lY,
               lWidth,
               lHeight,
               lMin,
               lMax,
               lValue,
               ulStyle,
               ulFillColor,
               ulBackgroundFillColor,
               ulOutlineColor,
               ulTextColor,
               ulBackgroundTextColor,
               pFont,
               pcText,
               pucImage,
               pucBackgroundImage,
               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 slider.

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

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

lWidth is the width of the slider.

lHeight is the height of the slider.

lMin is the minimum value for the slider (corresponding to the left or bottom position).

lMax is the maximum value for the slider (corresponding to the right or top position).

lValue is the initial value of the slider. This must lie in the range defined by *lMin* and *lMax*.

ulStyle is the style to be applied to the slider.

ulFillColor is the color used to fill in the slider.

ulBackgroundFillColor is the color used to fill in the background area of the slider.

ulOutlineColor is the color used to outline the slider.

ulTextColor is the color used to draw text on the slider.

ulBackgroundTextColor is the color used to draw text on the background portion of the slider.

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

pcText is a pointer to the text to draw on this slider.

pucImage is a pointer to the image to draw on this slider.

pucBackgroundImage is a pointer to the image to draw on the slider background.

pfnOnChange is a pointer to the function that is called to notify the application of slider value changes.

Description:

This macro provides an initialized slider 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:

- **SL_STYLE_OUTLINE** to indicate that the slider should be outlined.
- **SL_STYLE_FILL** to indicate that the slider should be filled.
- **SL_STYLE_BACKG_FILL** to indicate that the background portion of the slider should be filled.
- **SL_STYLE_TEXT** to indicate that the slider should have text drawn on its active portion (using *pFont* and *pcText*).
- **SL_STYLE_BACKG_TEXT** to indicate that the slider should have text drawn on its background portion (using *pFont* and *pcText*).
- **SL_STYLE_IMG** to indicate that the slider should have an image drawn on it (using *pucImage*).
- **SL_STYLE_BACKG_IMG** to indicate that the slider should have an image drawn on its background (using *pucBackgroundImage*).
- **SL_STYLE_TEXT_OPAQUE** to indicate that the slider text should be drawn opaque (in other words, drawing the background pixels).
- **SL_STYLE_BACKG_TEXT_OPAQUE** to indicate that the slider text should be drawn opaque in the background portion of the widget. (in other words, drawing the background pixels).
- **SL_STYLE_VERTICAL** to indicate that this is a vertical slider rather than a horizontal one (the default if this style flag is not set).

■ **SL_STYLE_LOCKED** to indicate that the slider is being used as an indicator and should ignore user input.

Returns:
Nothing; this is not a function.

12.2.3.13 SliderBackgroundFillOff

Disables filling of the background area of a slider widget.

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

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

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

Returns:
None.

12.2.3.14 SliderBackgroundFillOn

Enables filling of the background area of a slider widget.

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

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

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

Returns:
None.

12.2.3.15 SliderBackgroundImageOff

Disables the image on the background area of a slider widget.

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

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

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

Returns:
None.

12.2.3.16 SliderBackgroundImageOn

Enables the image on the background area of a slider widget.

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

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

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

Returns:
None.

12.2.3.17 SliderBackgroundImageSet

Changes the image drawn on the background area of a slider widget.

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

Parameters:
pWidget is a pointer to the slider widget to be modified.
pImg is a pointer to the image to draw onto the background area of the slider.

Description:
This function changes the image that is drawn onto the background area of the slider. This image will be centered within the widget rectangle and the portion in the area not represented by the current slider value will be visible. The display is not updated until the next paint request.

Returns:
None.

12.2.3.18 SliderBackgroundTextColorSet

Sets the background text color of a slider widget.

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

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

Description:
    This function changes the color used to draw text on the slider's background portion on the display. The display is not updated until the next paint request.

Returns:
    None.

12.2.3.19 SliderBackgroundTextOff

Disables the text on the background portion of a slider widget.

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

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

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

Returns:
    None.

12.2.3.20 SliderBackgroundTextOn

Enables the text on the background portion of a slider widget.

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

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

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

Returns:
    None.

12.2.3.21 SliderBackgroundTextOpaqueOff

Disables opaque background text on a slider widget.

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

Parameters:
>*pWidget* is a pointer to the slider widget to modify.

Description:
>This function disables the use of opaque text on the background portion of this slider. 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 slider image) to show through the text. Note that SL_STYLE_BACKG_TEXT must also be cleared to disable text rendering on the slider background area.

Returns:
>None.

12.2.3.22 SliderBackgroundTextOpaqueOn

Enables opaque background text on a slider widget.

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

Parameters:
>*pWidget* is a pointer to the slider widget to modify.

Description:
>This function enables the use of opaque text on the background portion of this slider. When using opaque text, both the foreground and background pixels of the text are drawn on the screen, blocking out the previously drawn pixels. Note that SL_STYLE_BACKG_TEXT must also be set to enable text rendering on the slider background area.

Returns:
>None.

12.2.3.23 SliderCallbackSet

Sets the function to call when this slider widget's value changes.

Definition:
```
#define SliderCallbackSet(pWidget,
                          pfnCallback)
```

Parameters:
>*pWidget* is a pointer to the slider widget to modify.
>*pfnCallback* is a pointer to the function to call.

Description:
This function sets the function to be called when the value represented by the slider changes.

Returns:
None.

12.2.3.24 SliderFillColorBackgroundedSet

Sets the fill color for the background area of a slider widget.

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

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

Description:
This function changes the color used to fill the background area of the slider on the display. The display is not updated until the next paint request.

Returns:
None.

12.2.3.25 SliderFillColorSet

Sets the fill color for the active area of a slider widget.

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

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

Description:
This function changes the color used to fill the active are of the slider on the display. The display is not updated until the next paint request.

Returns:
None.

12.2.3.26 SliderFillOff

Disables filling of the active area of a slider widget.

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

Parameters:
>   *pWidget* is a pointer to the slider widget to modify.

Description:
>   This function disables the filling of the active area of a slider widget. The display is not updated until the next paint request.

Returns:
>   None.

12.2.3.27 SliderFillOn

Enables filling of the active area of a slider widget.

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

Parameters:
>   *pWidget* is a pointer to the slider widget to modify.

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

Returns:
>   None.

12.2.3.28 SliderFontSet

Sets the font for a slider widget.

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

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

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

Returns:
>   None.

12.2.3.29 SliderImageOff

Disables the image on the active area of a slider widget.

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

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

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

Returns:
None.

12.2.3.30 SliderImageOn

Enables the image on the active area of a slider widget.

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

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

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

Returns:
None.

12.2.3.31 SliderImageSet

Changes the image drawn on the active area of a slider widget.

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

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

Description:
This function changes the image that is drawn on the active area of the slider. This image will be centered within the widget rectangle and the portion represented by the current slider value will be visible. The display is not updated until the next paint request.

Returns:
None.

12.2.3.32 SliderLock

Locks a slider making it ignore pointer input.

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

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

Description:
This function locks a slider widget and makes it ignore all pointer input. When locked, a slider acts as a passive indicator. Its value may be changed using SliderValueSet() and the value display updated using WidgetPaint() but no user interaction via the pointer will change the widget value.

Returns:
None.

12.2.3.33 SliderOutlineColorSet

Sets the outline color of a slider widget.

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

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

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

Returns:
None.

12.2.3.34 SliderOutlineOff

Disables outlining of a slider widget.

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

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

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

Returns:
　　None.

12.2.3.35 SliderOutlineOn

Enables outlining of a slider widget.

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

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

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

Returns:
　　None.

12.2.3.36 SliderRangeSet

Changes the value range for a slider widget.

Definition:
```
#define SliderRangeSet(pWidget,
                       lMinimum,
                       lMaximum)
```

Parameters:
　　pWidget is a pointer to the slider widget to be modified.
　　lMinimum is the minimum value that the slider will report.
　　lMaximum is the maximum value that the slider will report.

Description:
　　This function changes the range of a slider. Slider positions are reported in terms of this range with the current position of the slider on the display being scaled and translated into this range such that the minimum value represents the left position of a horizontal slider or the bottom position of a vertical slider and the maximum value represents the other end of the slider range. Note that this function does not cause the slider to be redrawn. The caller must call WidgetPaint() explicitly after this call to ensure that the widget is redrawn.

Returns:
　　None.

12.2.3.37 SliderStruct

Declares an initialized slider widget data structure.

Definition:

```
#define SliderStruct(pParent,
                     pNext,
                     pChild,
                     pDisplay,
                     lX,
                     lY,
                     lWidth,
                     lHeight,
                     lMin,
                     lMax,
                     lValue,
                     ulStyle,
                     ulFillColor,
                     ulBackgroundFillColor,
                     ulOutlineColor,
                     ulTextColor,
                     ulBackgroundTextColor,
                     pFont,
                     pcText,
                     pucImage,
                     pucBackgroundImage,
                     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 slider.
>
> *lX* is the X coordinate of the upper left corner of the slider.
>
> *lY* is the Y coordinate of the upper left corner of the slider.
>
> *lWidth* is the width of the slider.
>
> *lHeight* is the height of the slider.
>
> *lMin* is the minimum value for the slider (corresponding to the left or bottom position).
>
> *lMax* is the maximum value for the slider (corresponding to the right or top position).
>
> *lValue* is the initial value of the slider. This must lie in the range defined by *lMin* and *lMax*.
>
> *ulStyle* is the style to be applied to the slider.
>
> *ulFillColor* is the color used to fill in the slider.
>
> *ulBackgroundFillColor* is the color used to fill the background area of the slider.
>
> *ulOutlineColor* is the color used to outline the slider.
>
> *ulTextColor* is the color used to draw text on the slider.
>
> *ulBackgroundTextColor* is the color used to draw text on the background portion of the slider.
>
> *pFont* is a pointer to the font to be used to draw text on the slider.
>
> *pcText* is a pointer to the text to draw on this slider.
>
> *pucImage* is a pointer to the image to draw on this slider.
>
> *pucBackgroundImage* is a pointer to the image to draw on the slider background.
>
> *pfnOnChange* is a pointer to the function that is called to notify the application of slider value changes.

Description:

This macro provides an initialized slider 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:

```
tSliderWidget g_sSlider = SliderStruct(...);
```

Or, in an array of variables:

```
tSliderWidget g_psSliders[] =
{
    SliderStruct(...),
    SliderStruct(...)
};
```

ulStyle is the logical OR of the following:

- **SL_STYLE_OUTLINE** to indicate that the slider should be outlined.
- **SL_STYLE_FILL** to indicate that the slider should be filled.
- **SL_STYLE_BACKG_FILL** to indicate that the background portion of the slider should be filled.
- **SL_STYLE_TEXT** to indicate that the slider should have text drawn on its active portion (using *pFont* and *pcText*).
- **SL_STYLE_BACKG_TEXT** to indicate that the slider should have text drawn on its background portion (using *pFont* and *pcText*).
- **SL_STYLE_IMG** to indicate that the slider should have an image drawn on it (using *pucImage*).
- **SL_STYLE_BACKG_IMG** to indicate that the slider should have an image drawn on its background (using *pucBackgroundImage*).
- **SL_STYLE_TEXT_OPAQUE** to indicate that the slider text should be drawn opaque (in other words, drawing the background pixels).
- **SL_STYLE_BACKG_TEXT_OPAQUE** to indicate that the slider text should be drawn opaque in the background portion of the widget. (in other words, drawing the background pixels).
- **SL_STYLE_VERTICAL** to indicate that this is a vertical slider rather than a horizontal one (the default if this style flag is not set).
- **SL_STYLE_LOCKED** to indicate that the slider is being used as an indicator and should ignore user input.

Returns:

Nothing; this is not a function.

12.2.3.38 SliderTextColorSet

Sets the text color of the active portion of a slider widget.

Definition:

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

Parameters:

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

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

Description:
This function changes the color used to draw text on the active portion of the slider on the display. The display is not updated until the next paint request.

Returns:
None.

12.2.3.39 SliderTextOff

Disables the text on the active portion of a slider widget.

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

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

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

Returns:
None.

12.2.3.40 SliderTextOn

Enables the text on the active portion of a slider widget.

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

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

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

Returns:
None.

12.2.3.41 SliderTextOpaqueOff

Disables opaque text on the active portion of a slider widget.

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

Parameters:
> **pWidget** is a pointer to the slider widget to modify.

Description:
> This function disables the use of opaque text on the active portion of this slider. 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 slider image) to show through the text. Note that SL_STYLE_TEXT must also be cleared to disable text rendering on the slider active area.

Returns:
> None.

12.2.3.42 SliderTextOpaqueOn

Enables opaque text on the active portion of a slider widget.

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

Parameters:
> **pWidget** is a pointer to the slider widget to modify.

Description:
> This function enables the use of opaque text on the active portion of this slider. When using opaque text, both the foreground and background pixels of the text are drawn on the screen, blocking out the previously drawn pixels. Note that SL_STYLE_TEXT must also be set to enable text rendering on the slider active area.

Returns:
> None.

12.2.3.43 SliderTextSet

Changes the text drawn on a slider widget.

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

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

Description:
> This function changes the text that is drawn onto the slider. The string is centered across the slider and straddles the active and background portions of the widget. The display is not updated until the next paint request.

Returns:
> None.

12.2.3.44 SliderUnlock

Unlocks a slider making it pay attention to pointer input.

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

Parameters:
> ***pWidget*** is a pointer to the slider widget to modify.

Description:
> This function unlocks a slider widget. When unlocked, a slider will respond to pointer input by setting its value appropriately and informing the application via callbacks.

Returns:
> None.

12.2.3.45 SliderValueSet

Changes the minimum value for a slider widget.

Definition:
```
#define SliderValueSet(pWidget,
                       lVal)
```

Parameters:
> ***pWidget*** is a pointer to the slider widget to be modified.
> ***lVal*** is the new value to set for the slider. This is in terms of the value range currently set for the slider.

Description:
> This function changes the value that the slider will display the next time the widget is painted. The caller is responsible for ensuring that the value passed is within the range specified for the target widget. The caller must call WidgetPaint() explicitly after this call to ensure that the widget is redrawn.

Returns:
> None.

12.2.3.46 SliderVerticalSet

Sets the vertical or horizontal style for a slider widget

Definition:
```
#define SliderVerticalSet(pWidget,
                          bVertical)
```

Parameters:
> ***pWidget*** is a pointer to the slider widget to be modified.
> ***bVertical*** is **true** to set the vertical slider style or **false** to set the horizontal slider style.

Description:

This function allows the vertical or horizontal style to be set when creating slider widgets dynamically. The function will typically be called before the slider is first attached to the active widget tree. Since the vertical or horizontal style is intimately linked with the slider size and position on the display, it seldom makes sense to change this style for a widget which is already on the display.

Returns:

None.

12.2.4 Function Documentation

12.2.4.1 SliderInit

Initializes a slider widget.

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

Parameters:

pWidget is a pointer to the slider widget to initialize.

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

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

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

lWidth is the width of the slider.

lHeight is the height of the slider.

Description:

This function initializes the provided slider widget.

Returns:

None.

12.2.4.2 SliderMsgProc

Handles messages for a slider widget.

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

Parameters:

 pWidget is a pointer to the slider 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 slider 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.

13   Utilities

13.1   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.

13.2   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 includes 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>] [-w <num>] [-n] [-p <num>] -e <num>] <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.

`-w <num>`	Encodes the specified character index as a space regardless of the character which may be present in the font at that location. This is helpful in allowing a space to be included in a font which only encodes a subset of the characters which would not normally include the space character (for example, numeric digits only). If absent, this value defaults to 32, ensuring that character 32 is always the space.
`-n`	Overrides -w and causes no character to be encoded as a space unless the source font already contains a space.
`-p <num>`	Specifies the index of the first character in the font that is to be encoded. If the value is not provided, it defaults to 32 which is typically the space character.
`-e <num>`	Specifies the index of the last character in the font that is to be encoded. If the value is not provided, it defaults to 126 which, in ISO8859-1 is tilde ( ).
`<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.

The following would render a Computer Modern small-caps font at 44 points and generate an output font containing only characters 48 through 58 (the numeric digits). Additionally, character 47 in the encoded font (which is the first character and will be the character displayed if an attempt is made to render a character which is not included in the font) is forced to be a space:

```
ftrasterize -f cmscdigits -s 44 -w 47 -p 47 -e 58 cmcsc10.pfb
```

The output will be written to fontcmscdigits44.c and contain a definition for g_sFontCmscdigits44.

This application is located in `tools/ftrasterize`.

13.3   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 `tools/pnmtoc/lmi-button.scm`.

13.4   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.

To take a JPEG and convert it for use by the graphics library (using GIMP; a similar technique would be used in other graphics programs):

1. Load the file (File->Open).
2. Convert the image to indexed mode (Image->Mode->Indexed). Select "Generate optimum palette" and select either 2, 16, or 256 as the maximum number of colors (for a 1 BPP, 4 BPP, or 8 BPP image respectively). If the image is already in indexed mode, it can be converted to RGB mode (Image->Mode->RGB) and then back to indexed mode.
3. Save the file as a PNM image (File->Save As). Select raw format when prompted.
4. Use pnmtoc to convert the PNM image into a C array.

This sequence will be the same for any source image type (GIF, BMP, TIFF, and so on); once loaded into GIMP, it will treat all image types equally. For some source images, such as a GIF which is naturally an indexed format with 256 colors, the second step could be skipped if an 8 BPP image is desired in the application.

This application is located in `tools/pnmtoc`.

13.5   mkstringtable

The mkstringtable utility converts a comma separated file (.csv) to a table of strings that can be used by the graphics library. The source .csv file has a simple fixed format that supports multiple strings in multiple languages. The mkstringtable utility creates a .c and a .h file that can be compiled in with an application and used with the graphics library's string table handling functions. The mkstringtable utility will also attempt to compress the strings in the table in a way that allows the grpahics library string table functions to automatically decompress them when they are requested from the string table.

The mkstringtable utility can be run from the command line or as part of a build using a Makefile and has the following usage:

```
mkstringtable <csvfile> <rootname>
```

Where the arguments mean:

`<csvfile>`   Specifies the input .csv file to use to create a string table.

`<rootname>`   Specifies the root name of the output files as <rootname>.c and <root-name>.h. The value is also used in the naming of the string table variable which has a prototype in the <rootname>.h.

The format of the input .csv file is simple and easily edited in any plain text editor or a spreadsheet editor capable of reading and editing a .csv file. The .csv file format has a header row where the first entry in the row can be any string as it is ignored. The remaining entries in the row must be one of the GrLang∗ language definitions defined by the graphics library in the in grlib.h or they must have a definition that is valid for the application as this text is used directly in the C output file that is produced. Adding additional languages only requires that the value is unique in the table and that the name used is defined by the application.

Example: .csv file header for English(US), German, Spanish(SP), Italian

```
LanguageIDs,GrLangEnUS,GrLangDE,GrLangEsSP,GrLangIt
```

The strings are specified one per line in the .csv file. The first entry in any line is the value that is used as the actual text for the definition for the given string. The remaining entries should be the strings for each language specified in the header. Single words with no special characters do not require quotations, however any strings with a (,) character must be quoted as the (,) character is the delimiter for each item in the line. If the string has a quote character (") it must be preceded by another quote character. In the example below STR_QUOTE would result in the following strings:

```
Introduction in "English"
Einfuhrung, in Deutsch
Prueba
Verifica
```

Example: String definitions in a .csv file.

```
STR_CONFIG,Configuration,Konfigurieren,Configuracion,Configurazione
STR_INTRO,Introduction,Einfuhrung,Introduccion,Introduzione
STR_QUOTE,Introduction in ""English"","Einfuhrung, in Deutsch",Prueba,Verifica
...
```

The code that uses the string table produced by the mkstringtable application must refer to the strings by their identifier in the original .csv file. In the examples above, this means that the value STR_CONFIG would refer to the "Configuration" string in English(GrLangEnUS) or "Konfigurieren" in German(GrLangDE).

The resulting .c file contains the string table that must be included with the application that is using the string table. While the contents of this .c file are readable, the string table itself may be unintelligible due to the compression used on the strings themselves. The .h file that is created has the definition for the string table as well as an enumerated type "enum SCOMP_STR_INDEX" that contains all of the string indexes that were present in the original .csv file.

The graphics library's string table support functions directly access the string table based on the current language and string index. These string table support functions are the following: GrStringTableSet(), GrStringLanguageSet() and GrStringGet(). When referring to strings in the table, these functions should all use values that were included in the header file produced by the mkstringtable application. The GrStringTableSet() allows the application to use more than one string table if the application requires more languages or for any other purpose that the application needs.

Note: Only one string table is valid at any time as there is no support for multiple string tables being active at the same time.

Example: Setting the current string table and language.

```
//
// Set the string table.
//
GrStringTableSet(pucTablestrings);

//
// Set the current language to English(US).
//
GrLanguageSet(GrLangEnUS);
```

The GrStringGet() function handles retrieving strings from the string table and it uses the values in the first column of the CSV file as the index reference for a given string. The following example returns the string associated with the STR_CONFIG value in the current language set by the GrLanguageSet() function.
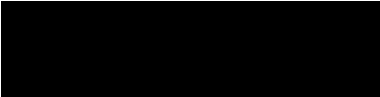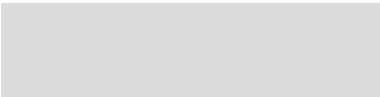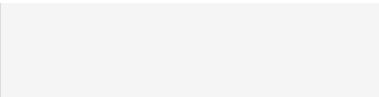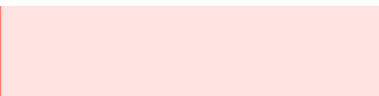
Example: Retrieving a string from the table.

```
//
// Retrieve the configuration string from the table.
//
GrStringGet(STR_CONFIG, pcData, sizeof(pcData));
```

14   Predefined Color Reference

There are a set of predefined colors in `grlib.h` that can be used for drawing on the screen. Other colors can be used by specifying their RGB values; the predefined colors are provided as a convenience.

The following table lists the predefined colors, along with a small color swatch showing the color.

ClrBlack	ClrDimGray	ClrGray
ClrDarkGray	ClrSilver	ClrLightGrey
ClrGainsboro	ClrWhiteSmoke	ClrWhite
ClrRosyBrown	ClrIndianRed	ClrBrown
ClrFireBrick	ClrLightCoral	ClrMaroon
ClrDarkRed	ClrRed	ClrSnow
ClrSalmon	ClrMistyRose	ClrTomato
ClrDarkSalmon	ClrOrangeRed	ClrCoral
ClrLightSalmon	ClrSienna	ClrChocolate

ClrSaddleBrown	ClrSeashell	ClrSandyBrown
ClrPeachPuff	ClrPeru	ClrLinen
ClrDarkOrange	ClrBisque	ClrBurlyWood
ClrTan	ClrAntiqueWhite	ClrNavajoWhite
ClrBlanchedAlmond	ClrPapayaWhip	ClrOrange
ClrMoccasin	ClrWheat	ClrOldLace
ClrFloralWhite	ClrGoldenrod	ClrDarkGoldenrod
ClrCornsilk	ClrGold	ClrLemonChiffon
ClrPaleGoldenrod	ClrKhaki	ClrDarkKhaki
ClrBeige	ClrLightGoldenrodYellow	ClrOlive

ClrYellow	ClrLightYellow	ClrIvory
ClrOliveDrab	ClrYellowGreen	ClrDarkOliveGreen
ClrGreenYellow	ClrLawnGreen	ClrChartreuse
ClrDarkSeaGreen	ClrForestGreen	ClrLimeGreen
ClrLightGreen	ClrPaleGreen	ClrDarkGreen
ClrGreen	ClrLime	ClrHoneydew
ClrMediumSeaGreen	ClrSeaGreen	ClrSpringGreen
ClrMintCream	ClrMediumSpringGreen	ClrMediumAquamarine
ClrAquamarine	ClrTurquoise	ClrLightSeaGreen
ClrMediumTurquoise	ClrDarkSlateGray	ClrPaleTurquoise

ClrTeal	ClrDarkCyan	ClrDarkTurquoise
ClrAqua	ClrCyan	ClrLightCyan
ClrAzure	ClrCadetBlue	ClrPowderBlue
ClrLightBlue	ClrDeepSkyBlue	ClrSkyBlue
ClrLightSkyBlue	ClrSteelBlue	ClrAliceBlue
ClrDodgerBlue	ClrSlateGray	ClrLightSlateGray
ClrLightSteelBlue	ClrCornflowerBlue	ClrRoyalBlue
ClrMidnightBlue	ClrLavender	ClrNavy
ClrDarkBlue	ClrMediumBlue	ClrBlue
ClrGhostWhite	ClrDarkSlateBlue	ClrSlateBlue

ClrMediumSlateBlue	ClrMediumPurple	ClrBlueViolet
ClrIndigo	ClrDarkOrchid	ClrDarkViolet
ClrMediumOrchid	ClrThistle	ClrPlum
ClrViolet	ClrPurple	ClrDarkMagenta
ClrFuchsia	ClrMagenta	ClrOrchid
ClrMediumVioletRed	ClrDeepPink	ClrHotPink
ClrLavenderBlush	ClrPaleVioletRed	ClrCrimson
ClrPink	ClrLightPink	

15   Font Reference

There are a large range of fonts supplied with the Graphics Library that can be used for rendering text on the screen. Additional fonts can be created by using the `ftrasterize` utility to compress font files into the format required by the Graphics Library.

The following table lists the supplied fonts (using the name of the global variable that contains the font), along with a string rendered using that font.

g_sFontFixed6x8

The quick brown fox jumps over the lazy dog.

g_sFontCm12

The quick brown fox jumps over the lazy dog.

g_sFontCm12b

The quick brown fox jumps over the lazy dog.

g_sFontCm12i

The quick brown fox jumps over the lazy dog.

g_sFontCmsc12

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss12

The quick brown fox jumps over the lazy dog.

g_sFontCmss12b

The quick brown fox jumps over the lazy dog.

g_sFontCmss12i

The quick brown fox jumps over the lazy dog

g_sFontCmtt12

The quick brown fox jumps over the lazy dog.

g_sFontCm14

The quick brown fox jumps over the lazy dog.

g_sFontCm14b

The quick brown fox jumps over the lazy dog.

g_sFontCm14i

The quick brown fox jumps over the lazy dog.

g_sFontCmsc14

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

g_sFontCmss14

The quick brown fox jumps over the lazy dog.

g_sFontCmss14b

The quick brown fox jumps over the lazy dog.

g_sFontCmss14i

The quick brown fox jumps over the lazy dog.

g_sFontCmtt14

The quick brown fox jumps over the lazy dog.

g_sFontCm16

The quick brown fox jumps over the lazy dog.

g_sFontCm16b

The quick brown fox jumps over the lazy dog.

g_sFontCm16i

The quick brown fox jumps over the lazy dog.

g_sFontCmsc16

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss16

The quick brown fox jumps over the lazy dog.

g_sFontCmss16b

The quick brown fox jumps over the lazy dog.

g_sFontCmss16i

The quick brown fox jumps over the lazy dog.

g_sFontCmtt16

The quick brown fox jumps over the lazy dog.

g_sFontCm18

The quick brown fox jumps over the lazy dog.

g_sFontCm18b

The quick brown fox jumps over the lazy dog.

g_sFontCm18i

The quick brown fox jumps over the lazy dog.

g_sFontCmsc18

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss18

The quick brown fox jumps over the lazy dog

g_sFontCmss18b

The quick brown fox jumps over the lazy dog.

g_sFontCmss18i
> *The quick brown fox jumps over the lazy dog.*

g_sFontCmtt18
> The quick brown fox jumps over the lazy dog.

g_sFontCm20
> The quick brown fox jumps over the lazy dog.

g_sFontCm20b
> **The quick brown fox jumps over the lazy dog.**

g_sFontCm20i
> *The quick brown fox jumps over the lazy dog.*

g_sFontCmsc20
> THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss20
> The quick brown fox jumps over the lazy dog.

g_sFontCmss20b
> **The quick brown fox jumps over the lazy dog.**

g_sFontCmss20i
> *The quick brown fox jumps over the lazy dog.*

g_sFontCmtt20
> The quick brown fox jumps over the lazy dog.

g_sFontCm22
> The quick brown fox jumps over the lazy dog.

g_sFontCm22b
> **The quick brown fox jumps over the lazy dog.**

g_sFontCm22i
> *The quick brown fox jumps over the lazy dog.*

g_sFontCmsc22
> THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss22
> The quick brown fox jumps over the lazy dog.

g_sFontCmss22b
> **The quick brown fox jumps over the lazy dog.**

g_sFontCmss22i
The quick brown fox jumps over the lazy dog.

g_sFontCmtt22
The quick brown fox jumps over the lazy dog.

g_sFontCm24
The quick brown fox jumps over the lazy dog.

g_sFontCm24b
The quick brown fox jumps over the lazy dog.

g_sFontCm24i
The quick brown fox jumps over the lazy dog.

g_sFontCmsc24
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss24
The quick brown fox jumps over the lazy dog.

g_sFontCmss24b
The quick brown fox jumps over the lazy dog.

g_sFontCmss24i
The quick brown fox jumps over the lazy dog.

g_sFontCmtt24
The quick brown fox jumps over the lazy dog.

g_sFontCm26
The quick brown fox jumps over the lazy dog.

g_sFontCm26b
The quick brown fox jumps over the lazy dog.

g_sFontCm26i
The quick brown fox jumps over the lazy dog.

g_sFontCmsc26
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss26
The quick brown fox jumps over the lazy dog.

g_sFontCmss26b
The quick brown fox jumps over the lazy dog.

g_sFontCmss26i
The quick brown fox jumps over the lazy dog.

g_sFontCmtt26
The quick brown fox jumps over the lazy dog.

g_sFontCm28
The quick brown fox jumps over the lazy dog.

g_sFontCm28b
The quick brown fox jumps over the lazy dog.

g_sFontCm28i
The quick brown fox jumps over the lazy dog.

g_sFontCmsc28
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss28
The quick brown fox jumps over the lazy dog.

g_sFontCmss28b
The quick brown fox jumps over the lazy dog.

g_sFontCmss28i
The quick brown fox jumps over the lazy dog.

g_sFontCmtt28
The quick brown fox jumps over the lazy dog.

g_sFontCm30
The quick brown fox jumps over the lazy dog.

g_sFontCm30b
The quick brown fox jumps over the lazy dog.

g_sFontCm30i
The quick brown fox jumps over the lazy dog.

g_sFontCmsc30
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss30
The quick brown fox jumps over the lazy dog.

g_sFontCmss30b
The quick brown fox jumps over the lazy dog.

g_sFontCmss30i
The quick brown fox jumps over the lazy dog.

g_sFontCmtt30
The quick brown fox jumps over the lazy dog.

g_sFontCm32
The quick brown fox jumps over the lazy dog.

g_sFontCm32b
The quick brown fox jumps over the lazy dog.

g_sFontCm32i
The quick brown fox jumps over the lazy dog.

g_sFontCmsc32
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss32
The quick brown fox jumps over the lazy dog.

g_sFontCmss32b
The quick brown fox jumps over the lazy dog.

g_sFontCmss32i
The quick brown fox jumps over the lazy dog.

g_sFontCmtt32
The quick brown fox jumps over the lazy dog

g_sFontCm34
The quick brown fox jumps over the lazy dog.

g_sFontCm34b
The quick brown fox jumps over the lazy dog.

g_sFontCm34i
The quick brown fox jumps over the lazy dog.

g_sFontCmsc34

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

g_sFontCmss34

The quick brown fox jumps over the lazy dog.

g_sFontCmss34b

The quick brown fox jumps over the lazy dog.

g_sFontCmss34i

The quick brown fox jumps over the lazy dog.

g_sFontCmtt34

The quick brown fox jumps over the lazy dog

g_sFontCm36

The quick brown fox jumps over the lazy dog.

g_sFontCm36b

The quick brown fox jumps over the lazy dog.

g_sFontCm36i

The quick brown fox jumps over the lazy dog.

g_sFontCmsc36

THE QUICK BROWN FOX JUMPS OVER THE LAZY I

g_sFontCmss36

The quick brown fox jumps over the lazy dog.

g_sFontCmss36b

The quick brown fox jumps over the lazy dog.

g_sFontCmss36i

The quick brown fox jumps over the lazy dog.

g_sFontCmtt36

The quick brown fox jumps over the lazy d

g_sFontCm38

The quick brown fox jumps over the lazy dog.

g_sFontCm38b

The quick brown fox jumps over the lazy dog

g_sFontCm38i

The quick brown fox jumps over the lazy dog.

g_sFontCmsc38

THE QUICK BROWN FOX JUMPS OVER THE LAZY

g_sFontCmss38

The quick brown fox jumps over the lazy dog.

g_sFontCmss38b

The quick brown fox jumps over the lazy dog.

g_sFontCmss38i

The quick brown fox jumps over the lazy dog.

g_sFontCmtt38

The quick brown fox jumps over the lazy d

g_sFontCm40

The quick brown fox jumps over the lazy dog

g_sFontCm40b

The quick brown fox jumps over the lazy

g_sFontCm40i

The quick brown fox jumps over the lazy dog

g_sFontCmsc40

THE QUICK BROWN FOX JUMPS OVER THE L

g_sFontCmss40

The quick brown fox jumps over the lazy dog.

g_sFontCmss40b

The quick brown fox jumps over the lazy d

g_sFontCmss40i

The quick brown fox jumps over the lazy do

g_sFontCmtt40

The quick brown fox jumps over the la

g_sFontCm42

The quick brown fox jumps over the lazy do

g_sFontCm42b

The quick brown fox jumps over the la

g_sFontCm42i

The quick brown fox jumps over the lazy

g_sFontCmsc42

THE QUICK BROWN FOX JUMPS OVER THE

g_sFontCmss42

The quick brown fox jumps over the lazy dog.

g_sFontCmss42b

The quick brown fox jumps over the lazy

g_sFontCmss42i

The quick brown fox jumps over the lazy d

g_sFontCmtt42

The quick brown fox jumps over the l

g_sFontCm44

The quick brown fox jumps over the lazy

g_sFontCm44b

The quick brown fox jumps over the l

g_sFontCm44i

The quick brown fox jumps over the laz

g_sFontCmsc44

THE QUICK BROWN FOX JUMPS OVER TH

g_sFontCmss44

The quick brown fox jumps over the lazy dog

g_sFontCmss44b

The quick brown fox jumps over the laz

g_sFontCmss44i

The quick brown fox jumps over the lazy

g_sFontCmtt44

The quick brown fox jumps over th

g_sFontCm46

The quick brown fox jumps over the lazy

g_sFontCm46b

The quick brown fox jumps over the

g_sFontCm46i

The quick brown fox jumps over the la

g_sFontCmsc46

THE QUICK BROWN FOX JUMPS OVER T

g_sFontCmss46

The quick brown fox jumps over the lazy d

g_sFontCmss46b

The quick brown fox jumps over the la

g_sFontCmss46i

The quick brown fox jumps over the la

g_sFontCmtt46

The quick brown fox jumps over th

g_sFontCm48

The quick brown fox jumps over the la

g_sFontCm48b

The quick brown fox jumps over the

g_sFontCm48i

The quick brown fox jumps over the l

g_sFontCmsc48

THE QUICK BROWN FOX JUMPS OVER

g_sFontCmss48

The quick brown fox jumps over the lazy

g_sFontCmss48b

The quick brown fox jumps over the

g_sFontCmss48i

The quick brown fox jumps over the l

g_sFontCmtt48

The quick brown fox jumps over t

IMPORTANT NOTICE