




# TivaWare™ Graphics Library

**USER'S GUIDE**

---

# Copyright

Copyright © 2008-2015 Texas Instruments Incorporated. All rights reserved. Tiva and TivaWare are trademarks of Texas Instruments Instruments. ARM and Thumb are registered trademarks and Cortex is a trademark of ARM Limited. Other names and brands may be claimed as the property of others.

 Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments  
108 Wild Basin, Suite 350  
Austin, TX 78746  
[www.ti.com/tiva-c](http://www.ti.com/tiva-c)



## Revision Information

This is version 2.1.2.111 of this document, last updated on December 16, 2015.

# Table of Contents

<b>Copyright</b> .....	<b>2</b>
<b>Revision Information</b> .....	<b>2</b>
<b>1 Introduction</b> .....	<b>5</b>
<b>2 Display Driver</b> .....	<b>9</b>
2.1 Introduction .....	9
2.2 Definitions .....	9
<b>3 Graphics Primitives</b> .....	<b>15</b>
3.1 Introduction .....	15
3.2 Fonts and Text Handling .....	19
3.3 Definitions .....	27
<b>4 Widget Framework</b> .....	<b>85</b>
4.1 Introduction .....	85
4.2 Definitions .....	85
<b>5 Canvas Widget</b> .....	<b>97</b>
5.1 Introduction .....	97
5.2 Definitions .....	97
<b>6 Checkbox Widget</b> .....	<b>115</b>
6.1 Introduction .....	115
6.2 Definitions .....	115
<b>7 Container Widget</b> .....	<b>131</b>
7.1 Introduction .....	131
7.2 Definitions .....	131
<b>8 Image Button Widget</b> .....	<b>145</b>
8.1 Introduction .....	145
8.2 Definitions .....	145
<b>9 ListBox Widget</b> .....	<b>163</b>
9.1 Introduction .....	163
9.2 Definitions .....	164
<b>10 Keyboard Widget</b> .....	<b>179</b>
10.1 Introduction .....	179
10.2 Definitions .....	180
<b>11 Push Button Widget</b> .....	<b>199</b>
11.1 Introduction .....	199
11.2 Definitions .....	199
<b>12 Radio Button Widget</b> .....	<b>221</b>
12.1 Introduction .....	221
12.2 Definitions .....	222
<b>13 Slider Widget</b> .....	<b>237</b>
13.1 Introduction .....	237
13.2 Definitions .....	238
<b>14 Utilities</b> .....	<b>261</b>
14.1 Introduction .....	261
14.2 ftrasterize .....	261
14.3 lmi-button .....	265

14.4 pnmtoC . . . . .	265
14.5 mkstringtable . . . . .	266
<b>15 Predefined Color Reference . . . . .</b>	<b>269</b>
<b>16 Font Reference . . . . .</b>	<b>275</b>
<b>IMPORTANT NOTICE . . . . .</b>	<b>430</b>

# 1 Introduction

The Texas Instruments® Graphics Library provides a set of graphics primitives and a widget set for creating graphical user interfaces on 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
- Mentor Graphics® Sourcery™ CodeBench
- IAR Embedded Workbench®
- 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.

<code>Makefile</code>	The rules for building the graphics library.
<code>canvas.c</code>	The source code for the canvas widget, which is described in chapter 5.
<code>canvas.h</code>	The header containing prototypes for the canvas widget.
<code>ccs/</code>	The directory containing the Code Composer Studio project files.
<code>charmap.c</code>	The source code for the text codepage mapping functions, which are described in chapter 3.
<code>checkbox.c</code>	The source code for the check box widget, which is described in chapter 6.
<code>checkbox.h</code>	The header containing prototypes for the checkbox widget.
<code>circle.c</code>	The source code for the circle primitives, which are described in chapter 3.
<code>container.c</code>	The source code for the container widget, which is described in chapter 7.
<code>container.h</code>	The header containing prototypes for the container widget.
<code>context.c</code>	The source code for the drawing context, which is described in chapter 3.
<code>fonts/</code>	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 16.
<code>ftrasterize/</code>	The source code for the ftrasterize program, which is described in chapter 14.
<code>grrlib.Opt</code>	The Keil uVision project options file used for building the graphics library.
<code>grrlib.Uv2</code>	The Keil uVision project file used for building the graphics library.
<code>grrlib.ewd</code>	The IAR Embedded Workbench project options file used for building the graphics library.
<code>grrlib.ewp</code>	The IAR Embedded Workbench project file used for building the graphics library.
<code>grrlib.h</code>	The header containing prototypes for the graphics primitives.
<code>image.c</code>	The source code for the image primitives, which are described in chapter 3.
<code>line.c</code>	The source code for the line primitives, which are described in chapter 3.
<code>offscribpp.c</code>	The source code for the 1 BPP off-screen buffer display driver, which is described in chapter 3.

<code>offscr4bpp.c</code>	The source code for the 4 BPP off-screen buffer display driver, which is described in chapter 3.
<code>offscr8bpp.c</code>	The source code for the 8 BPP off-screen buffer display driver, which is described in chapter 3.
<code>pnmtoc/</code>	The source code for the <code>pnmtoc</code> program, which is described in chapter 14.
<code>pushbutton.c</code>	The source code for the push button widget, which is described in chapter 11.
<code>pushbutton.h</code>	The header containing prototypes for the push button widget.
<code>radiobutton.c</code>	The source code for the radio button widget, which is described in chapter 12.
<code>radiobutton.h</code>	The header containing prototypes for the radio button widget.
<code>readme.txt</code>	A short file describing the highlights of the graphics library.
<code>rectangle.c</code>	The source code for the rectangle primitives, which are described in chapter 3.
<code>slider.c</code>	The source code for the slider widget, which is described in chapter 13.
<code>slider.h</code>	The header containing prototypes for the slider widget.
<code>string.c</code>	The source code for the string primitives, which are described in chapter 3.
<code>widget.c</code>	The source code for the widget framework, which is described in chapter 4.
<code>widget.h</code>	The header containing prototypes for the widget framework.





## 2 Display Driver

Introduction .....	9
Definitions .....	9

### 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 operations. 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 IX1, long IX2, long IY, unsigned long ulValue)
- void `LineDrawV` (void \*pvDisplayData, long IX, long IY1, long IY2, unsigned long ulValue)
- void `PixelDraw` (void \*pvDisplayData, long IX, long IY, unsigned long ulValue)
- void `PixelDrawMultiple` (void \*pvDisplayData, long IX, long IY, long IX0, long ICount, long IBPP, 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

Introduction .....	15
Fonts and Text Handling .....	19
Definitions .....	27

### 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 `gplib/gplib.h` that can be used if desired; a list of these colors and a corresponding color swatch can be found in chapter 15.

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 reflected onto the screen.

A large set of fonts are provided, based on the Computer Modern typeface defined by Professor Donald E. Knuth for the  $\text{T}_{\text{E}}\text{X}$  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 `gplib/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 16.

**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 `gplib/circle.c`, `gplib/context.c`, `gplib/image.c`, `gplib/line.c`, `gplib/offscrib.c`, `gplib/offscr4bpp.c`, `gplib/offscr8bpp.c`, `gplib/rectangle.c`, and `gplib/string.c`, with `gplib/gplib.h` containing the API declarations 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 of 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 14](#).

### 3.1.4 Font Glyph Data Format

The graphics library supports three distinct bitmap font formats offering a variety of features depending upon the needs of an application. All three formats encode individual characters in the same way but use different headers and character glyph lookup methods. Information on these formats can be found in [chapter 3.2](#).

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-existent character from a 14x8 font):

```
.....  
.....  
.....  
.....xx.....  
.....xxxx.....  
..xxxxxxxxx..  
..xxxxxxxxx..  
.....
```

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.

A utility (`ftrasterize`) is provided to assist in converting fonts into these formats; it is documented in chapter [14](#).

## 3.2 Fonts and Text Handling

### 3.2.1 Definitions

Understanding the graphics library text handling functions will be easier if the following terminology is known.

<code>ASCII</code>	American Standard Code for Information Interchange. ASCII is a 7 bit codepage with codepoints in the range 0x00 to 0x7F. It contains the basic upper- and lower-case Latin alphabet, numeric digits, common punctuation marks and terminal control codes. It is in common use in English-speaking countries but offers no way to encode accented characters or non-Latin alphabets.
<code>codepage</code>	A character encoding scheme mapping between codepoints and glyphs within a font. The codepage determines which character a given codepoint (character number) represents. For example, when using the ASCII codepage, codepoint 0x20 represents the space character.
<code>codepoint</code>	A single entry in a codepage. A number identifying a character in a font. Knowing the codepage in use, the codepoint (or character code) defines a single character.
<code>glyph</code>	A graphical representation of a single character in a font.
<code>font</code>	A collection of character glyphs in a particular typeface and size each represented by a codepoint.

ISO8859	A set of codepages containing characters identified by an 8 bit codepoint. Each ISO8859 variant identifies a different codepage. For example ISO8859-1 contains ASCII + Latin accented characters and ISO8859-5 contains ASCII + Cyrillic characters.
Unicode	An international standard defining a universal, 32 bit character encoding encompassing all writing systems. Unicode effectively does away with the complications associated with codepages by defining a unique code for every possible character in every world writing system.
UTF-8	Unicode Transformation Format (8). A variable length encoding system for Unicode text where any given character can be represented by 1 to 6 bytes depending upon the character. UTF-8 has the advantage that it is backwards compatible with ASCII and is commonly used in text file processing.
UTF-16	Unicode Transformation Format (16). A variable length encoding system for Unicode text where any given character can be represented by either one or two 16-bit words depending upon the character. UTF-16 is compatible with UCS-2 for codepoints in the 0-0xFFFF range. UTF-16 is also the native internal text format used by Windows (since Win2K), MacOS X and Java but text files typically do not use this format.

## 3.2.2 Font Formats

Four structures are used to describe a font to graphics library functions. Each of these share a common first field, `ucFormat`, allowing the graphics library font functions to accept a `tFont` pointer in all cases and to determine which underlying structure is in use and parse the data appropriately. The graphics library header and font headers under `third_party/fonts` define macros which provide `tFont *` pointers for all included fonts. This prevents users from having to include explicit pointer casts when passing font parameters to `glib` functions. It is, however, perfectly acceptable (and expected) to cast a `tFontEx`, `tFontWide` or `tFontWrapper` pointer to a `tFont` pointer for use with API functions that require a font parameter.

### 3.2.2.1 tFont

The `tFont` structure allows encoding of fonts containing the printable subset of the 7-bit ASCII encoding, characters 32 (space) to 126 (tilde). It offers the smallest font size for applications requiring use of the alphanumeric characters offered by ASCII. Since this encoding always contains 95 characters, applications which use only a subset of ASCII, for example the numeric digits, may find that the `tFontEx` structure offers a better alternative.

The header of a `tFont` structure contains information on the font character cell size and baseline along with a 96 entry glyph offset table and a pointer to an array of data containing the compressed or uncompressed font glyph data.

### 3.2.2.2 tFontEx

The `tFontEx` increases text rendering capabilities somewhat to allow the use of full 8 bit encodings rather than the 7 bit encodings supported by `tFont`. The structure allows encoding of any arbitrary

subset of characters with codepoints 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.

While this format can support any 8 bit character encoding, no codepage information is stored with the font itself so applications making use of such fonts must be careful to ensure that the source text rendered using the font makes use of the same codepage as the font that is rendering it. For example, if you have created an ISO8859-5 compatible font containing western and Cyrillic alphabets but pass it a string containing ISO8859-1 text with accented Latin characters, you will see incorrect characters rendered.

If your application intends to support multiple languages and either multiple source text codepages or an encoding standard such as UTF8, a `tFontEx` font may not offer the flexibility required and a `tFontWide` may be more appropriate. If supporting a subset of languages which can be rendered using one of the ISO8859 variants, however, `tFontEx` is a good choice since the additional header overhead compared to `tFont` is only 8 bytes.

### 3.2.2.3 tFontWide

The `tFontWide` structure allows use of international fonts with multiple alphabets and clean handling of source and font codepages. This format adds the ability to support multi-byte character encodings such as UTF8, UTF16 and UTF32 and removes the 95 or 256 character restrictions imposed by the `tFont` and `tFontEx` structures. This format of font is encoded without the need for absolute data pointers and so is suitable for use from non-linear-access storage via a wrapper module and the `tFontWrapper` structure.

Glyphs in a `tFontWide` font are encoded in blocks where a block contains a single, contiguous set of codepoints. Each block starts with its own glyph data offset table followed by the compressed or uncompressed data for each of the glyphs in the block.

A font is structured with a header indicating the structure format and cell size along with the codepage used by the font and the number of blocks the font contains. Following this is a table providing information on each of the blocks - start codepoint number, number of glyphs in the block and the offset to the block glyph table. The block glyph tables follow immediately after the block table with each block glyph table padded to a multiple of 4 bytes if necessary.

Graphically, the structure is as follows:

```

-----
0 | tFontWide header          |
  | ulNumBlocks = n         |
-----
  | tFontBlock[0]           | First block describes characters from
  | ulStartCodepoint = S0   | codepoint S0 to (S0 + N0 - 1). Offset
  | ulNumCodepoints = N0    | to glyph table is OF0 bytes from the
  | ulGlyphTableOffset = OF0 | start of the tFontWide header struct.
-----
  | ...                     |
-----
  | tFontBlock[n-1]        | Final block describes characters from
  | ulStartCodepoint = S(n-1) | codepoint S(n-1) to (S(n-1) + N(n-1)
  | ulNumCodepoints = N(n-1) | - 1).
  | ulGlyphTableOffset = OF(n-1) |
-----

```

Start of Block 0 Offset Table		
OF0	ulGlyphOffset = GO0	Offset from OF0 to glyph S0 data.
	ulGlyphOffset = GO1	Offset from OF0 to glyph (S0 + 1) data.
	...	
	ulGlyphOffset = GO(N0 - 1)	Offset from OF0 to glyph (S0 + N0 - 1)
Start of Block 0 Glyph Data		
GO0	Glyph data for codepoint S0 from block 0	
GO1	Glyph data for codepoint S0 + 1 from block 0	
	...	
GO (N-1)	Glyph data for codepoint (S0 + N0 - 1) from block 0	
	Padding to align OF1 on a 4 byte boundary (if necessary)	
Start of Block 1 Offset Table		
OF1	ulGlyphOffset = G10	Offset from OF1 to glyph S1 data.
	ulGlyphOffset = G11	Offset from OF1 to glyph (S1 + 1) data.
	...	
	ulGlyphOffset = G1(N1 - 1)	Offset from OF1 to glyph (S1 + N1 - 1)
Start of Block 1 Glyph Data		
G10	Glyph data for codepoint S1 from block 1	
G11	Glyph data for codepoint S1 + 1 from block 1	
	...	
G1 (N-1)	Glyph data for codepoint (S1 + N1 - 1) from block 1	
GO0	Glyph data for codepoint S0 from block 0	
Start of Block 2 Offset Table		
	...continued for remaining font blocks.	

Wide font files generated by the `ftrasterize` tool contain the structure defined as a constant array of unsigned characters but a pointer to this array can be cast to `(tFont *)` type and passed to any graphics library call requiring a font pointer. The graphics library headerfile, `glib.h`, and the various headers under `third_party/fonts` contain macro definitions for each font which provide this cast for you.

### 3.2.2.4 tFontWrapper

The `tFontWrapper` structure does not, technically define a font but it does provide a mechanism allowing applications to access `tFontWide` format binary fonts stored in memory which is not directly visible to the CPU such as in a file system or stored on a serial flash device. Like all other font structures, a `tFontWrapper` pointer may be cast to a `tFont` pointer and used with any graphics library function which expects a font pointer as a parameter.

Unlike the other font formats where the graphics library code parses the font data directly, a wrapped font is accessed via five function pointers stored in a `tFontAccessFuncs` structure within `tFontWrapper`. These functions mirror similar functions in the glib API and allow the graphics library code to query information about the wrapped font and extract data for particular glyphs.

Font wrapper modules will implement the five functions required to populate the `tFontAccessFuncs` table and two others, a `FontLoad` function whose prototype is wrapper-specific but which prepares a font for use and returns a pointer that glib will pass on all calls to the access functions, and a `FontUnload` function which tidies up and releases any resources associated with accessing the font (closes files, frees buffers, etc.). Applications making use of wrapped fonts call the wrapper's `FontLoad` function to initialize the font to be used then populate the three fields of their `tFontWrapper` structure. After this, the `tFontWrapper` pointer may be used exactly as any other font pointer, casting it to a `(const tFont *)` type before passing it to any graphics library function which requires a font pointer.

Since font wrappers are board and application specific, none are included in the graphics library itself. The `fontview` example application included with some software releases does, however, offer a simple FAT file system wrapper and illustrates how to use a font stored on an SDCard.

## 3.2.3 Codepage Handling

The term “codepage” refers to a mapping of character numbers as stored in a file or string to specific displayable characters. When an application is dealing with ASCII text alone, codepage issues seldom occur since most of the common codepages in use are backwards compatible with ASCII, including the ASCII character set at the same position in their codespace in each case. When non-Western languages are to be used, however, the situation becomes more complex and an understanding and awareness of codepages becomes critical to ensure that text is displayed as expected.

The graphics library supports the concept of both source text codepages and font codepages and supplies functions necessary to set the codepage in use for strings passed to GLib functions, to query the codepage supported by a font and to convert between the two. Older fonts encoded using the `tFont` and `tFontEx` formats do not contain information defining their codepages and are assumed to contain ISO8859-1 (a standard western European codepage containing ASCII and a collection of accented characters). Fonts encoded in `tFontWide` format, however, contain a codepage specifier allowing applications to generate fonts indexed using different codepages and have these correctly handled when rendering strings encoded in a different codepage.

Source text codepage and encoding information can be set at two levels. A global default setting which will be used in all graphics contexts created by the application can be provided in a call to [GrLibInit\(\)](#). Alternatively, an application can set or change the source codepage at the context level using [GrStringCodepageSet\(\)](#). Note that the source codepage definitions actually include several “codepages” that are not strictly codepages at all but are encoding formats for Unicode text. UTF-8 (identified by `CODEPAGE_UTF_8`) and UTF-16 (`CODEPAGE_UTF_16`) are popular, variable length encoding methods supported by many editors. Pure Unicode `CODEPAGE_UNICODE` using a 32 bit character encoding is supported but is inefficient compared to UTF-8 and UTF-16 when encoding most text strings.

To allow the correct font glyphs to be displayed for each source character, applications must also provide GrLib with an array defining functions that can be used to convert between the chosen source text codepage or encoding and the codepage used by the chosen font. This is accomplished by means of the codepage mapping table, each entry of which defines a source and destination codepage and contains a pointer to the function used to convert a codepoint in the source codepage to one in the destination codepage. The function [GrCodepageMapTableSet\(\)](#) allows the codepage mapping table to be set for a specific graphics context or, alternatively, a default table that will apply to all created contexts may be provided in a call to [GrLibInit\(\)](#). GrLib exports several codepage mapping functions for common codepages (UTF-8, UTF32 and ISO8859 variants 1 through 5) so these may be used to populate the mapping table if your application will be making use of one of these codepages.

Although it may seem cumbersome to have the application generate and provide this mapping table when GrLib already has knowledge of the common codepage conversions, it makes sense for two main reasons. Firstly, an application which does not need to use a particular codepage may leave its conversion function out of the table and avoid the sometimes significant code overhead of including the conversion function and secondly, it allows the possibility of application- specific custom code pages and fonts.

The codepage mapping function that is used is set each time the application changes either the font attached to a context or the source codepage in use so the mapping table array need only be provided to GrLib once during initialization.

### 3.2.4 Language Rendering Hooks

The graphics library provides support for rendering characters from international character sets but, in its shipped form, does not support the following language-specific features:

1. Rendering directions other than left-to-right
2. Combining diacritics
3. Ligatures
4. Codepoint decomposition/recomposition

Each of these limitations can, however, be solved by using a hook provided by the graphics library that allows the main string rendering function to be replaced. This function is called in response to any call to [GrStringDraw\(\)](#) or [GrStringDrawCentered\(\)](#) and can be set on a context-by-context basis. By default, the function `GrDefaultStringRenderer` is used and this supports left-to-right text rendering. Applications can, however, replace this function with one which complies with the text formatting rules required by their desired language. Low level functions are provided by the graphics library to allow characters to be extracted from source strings, glyph data to be extracted from fonts and individual glyphs to be written to a given position on the display and these may be used to simplify development of the replacement string renderer function.



### 3.2.4.1 Text Rendering Direction

Left-to-right text rendering is used by all western European languages and is also supported for Chinese, Japanese and Korean. Other languages, notably Hebrew and Arabic, require right-to-left text rendering with the ability to insert left-to-right strings within the base left-to-right text. While such languages could be rendered by reformatting the source text into display order and rendering it left-to-right, the base graphics library does not currently support the ability to render strings stored in reading order in these languages.

To support different rendering directions, a replacement string renderer could be modeled on [GrDefaultStringRenderer\(\)](#) but update the rendering coordinates for each character differently to provide right-to-left rather than left-to-right text.

### 3.2.4.2 Combining Diacritics

The graphics library supports only precomposed diacritical characters (accents) where a single, unique codepoint defines both the character and its diacritical mark or marks. Some text encodings support the concept of combining diacritics. In these cases, the accented character is encoded using two or more distinct codepoints, one for the base character and one or more following it indicating the diacritical marks which must be added on top of the basic glyph.

A replacement text renderer with knowledge of codepoints representing diacritical characters could be written to ensure that diacritics are correctly rendered on top of the preceding character rather than in their own character cell as would be the case when using the default string renderer.

### 3.2.4.3 Ligatures

Ligatures are characteristics of a glyph which change depending upon the glyphs which either follow or precede it. This is seen when using cursive fonts, for example, where the join position between characters may differ depending upon the characters that are adjacent to one another. It is also commonly seen in Arabic where the appearance of one character is frequently affected by other characters around it.

Common ligatures in various languages have precomposed Unicode characters defined for them so, using a font which encodes these characters, an application could use a custom codepage mapping function to extract the precomposed characters from the string passed to [GrStringDraw\(\)](#) or, alternatively, use a custom string rendering function which looked ahead in the string to determine which of a number of ligatures should be used.

### 3.2.4.4 Codepoint Composition or Decomposition

The default string render assumes that there is a 1-to-1 mapping between a character extracted from the string and a font glyph that represents it. This is typically true but in some cases, notably Korean Hangul, it is possible to decode the Unicode value for a single ideograph into several other codepoints representing pieces of the character to be drawn and then render these individually to display the intended character. Using this approach, a very large number of ideographs can be generated from a relatively small alphabet of symbols and, as a result, a great deal of font storage could be saved.

Such a system could be implemented by using a font containing the relevant partial character alphabet glyphs and replacing the default string rendering function with one which understood how

to decompose and recompose characters based on these glyphs.

### 3.2.5 Using Custom Fonts & Codepages

When working on applications which require support for multiple languages using different alphabets, the size of storage required to hold font glyphs can quickly become a concern. This problem is especially apparent when dealing with Chinese, Japanese or Korean (CJK) text where many thousand possible glyphs exist. To help overcome some of these problems, the Graphics Library and the `mkstringtable` and `ftrasterize` tools provide some helpful features to aid in minimizing the amount of storage required to store glyph data and string tables.

In many cases, an application will not need to be able to display any arbitrary character from a given alphabet but, instead, requires only to display a number of fixed strings in one of several languages. In this case, a custom font containing only the required character glyphs can be used, thus removing the need to store all possible characters many of which will never be used. Building such a font is made straightforward by the character map table feature of the `ftrasterize` tool and options provided by the `mkstringtable` tool.

To generate an application-specific custom font, first ensure that all strings the application will need to display are stored in a single string table `.csv` file then invoke the `mkstringtable` tool and include the `-t` option. This will create an additional output text file which contains a list of all codepoints used in the string table and this can, in turn, be passed to `ftrasterize` using its `-c` and `-r` parameters to tell the tool to generate a font containing only glyphs for those specific codepoints. This font can then be used in the application, knowing that all characters that are required will be available.

This method has the advantage that the strings in the string table are still stored using the same codepage as the original string table so, when viewed in a debugger, for example, the strings are still readable (assuming your debugger supports the codepage that you used to create your original strings). As a side effect, it is also easy to add other blocks of characters into the output font merely by editing the character map file and adding blocks of characters that you know you will need but which may not necessarily exist in the string table. For example, an application which needs to display label strings for various controls and needs to be able to display decimal digits within those controls could have a block added to the character map which includes all the decimal digit characters to ensure that they are available for use.

One downside of this approach, however, is that the fonts generated are typically rather larger than they could be. The `tFontWide` structure can encode multiple blocks of characters but characters within a given block must represent contiguous codepoints. In a sparse string table, and especially when using ideographic alphabets such as Chinese, Japanese and Korean, this approach usually means you end up with a large number of single character blocks in the font and this has two problems. Firstly, there is a 12 byte overhead associated with each block definition and secondly the code required to find a given glyph in the font must do a lot more searching since each individual block header must be read to determine whether a given character falls into that block or not.

To prevent these problems, a second option can be used when creating the custom font and string table. This option, known as codepage remapping, recodes the string table such that the characters stored use a single, contiguous block of codepoints. Further, the codepoints are set up such that the most frequently used characters appear first in the new codepage and this, coupled with a variable length encoding scheme based on UTF-8, allows the strings to be stored very efficiently.

To make use of codepage remapping, substitute the `-t` switch passed to `mkstringtable.c` with `-r` and add the `-c` switch to select an identifier for the custom codepage. This identifier must be in the range `0x8000 - 0xFFFF` set aside in the graphics library header file, `glib.h`, as the range available for application use. This will encode the string table using the new custom codepage and also

write a tag into the output character map file to tell `ftrasterize` to do the same when it generates the custom font. When using the character map file with `ftrasterize`, be careful to include the “-z” switch and pass the same custom codepage number as you passed to `mkstringtable`. After running the two tools, you will end up with string table and font source files and a string table header file. The string table header contains some helpful macros that can be used when initializing the graphics library to use the custom font and string table.

If using codepage remapping, remember that the numbers used to represent the characters in your strings no longer conform to any standard codepage so you will not be able to read the string if viewed in a debugger memory window and you will not be able to make use of functions such as `usprintf` to format text for display using the new font since no remapping from ASCII, Unicode or ISO8859 to the custom codepage is supplied. As a result, codepage remapping is most useful when an application needs to display ONLY a set of fixed strings and will never need to manipulate arbitrary strings for display. If some string manipulation is required, tricks can be performed by encoding partial strings into the string table and constructing an output string by piecing these together via `usprintf()` as long as only the “s” formatting insert is used.

For a practical example of using custom fonts both with and without codepage remapping, see the “`lang_demo`” example application included with some software releases. The custom font used by these applications and a Makefile showing the steps necessary to create it and the associated string table can be found in the `third_party/fonts/lang_demo` directory of your StellarisWare installation.

## 3.3 Definitions

### Data Structures

- [tCodePointMap](#)
- [tContext](#)
- [tDisplay](#)
- [tFont](#)
- [tFontAccessFuncs](#)
- [tFontEx](#)
- [tFontWide](#)
- [tFontWrapper](#)
- [tGrLibDefaults](#)
- [tRectangle](#)

### Defines

- [CODEPAGE\\_ISO8859\\_1](#)
- [DpyColorTranslate](#)(psDisplay, ui32Value)
- [DpyFlush](#)(psDisplay)
- [DpyHeightGet](#)(psDisplay)
- [DpyLineDrawH](#)(psDisplay, i32X1, i32X2, i32Y, ui32Value)
- [DpyLineDrawV](#)(psDisplay, i32X, i32Y1, i32Y2, ui32Value)
- [DpyPixelDraw](#)(psDisplay, i32X, i32Y, ui32Value)
- [DpyPixelDrawMultiple](#)(psDisplay, i32X, i32Y, i32X0, i32Count, i32BPP, pui8Data, pui8Palette)

- [DpyRectFill](#)(psDisplay, psRect, ui32Value)
- [DpyWidthGet](#)(psDisplay)
- [FONT\\_EX\\_MARKER](#)
- [FONT\\_FMT\\_EX\\_PIXEL\\_RLE](#)
- [FONT\\_FMT\\_EX\\_UNCOMPRESSED](#)
- [FONT\\_FMT\\_PIXEL\\_RLE](#)
- [FONT\\_FMT\\_UNCOMPRESSED](#)
- [FONT\\_FMT\\_WIDE\\_PIXEL\\_RLE](#)
- [FONT\\_FMT\\_WIDE\\_UNCOMPRESSED](#)
- [FONT\\_FMT\\_WRAPPED](#)
- [FONT\\_WIDE\\_MARKER](#)
- [GrContextBackgroundSet](#)(psContext, ui32Value)
- [GrContextBackgroundSetTranslated](#)(psContext, ui32Value)
- [GrContextDpyHeightGet](#)(psContext)
- [GrContextDpyWidthGet](#)(psContext)
- [GrContextForegroundSet](#)(psContext, ui32Value)
- [GrContextForegroundSetTranslated](#)(psContext, ui32Value)
- [GrFlush](#)(psContext)
- [GrImageColorsGet](#)(pui8Image)
- [GrImageHeightGet](#)(pui8Image)
- [GrImageWidthGet](#)(pui8Image)
- [GRLIB\\_DRIVER\\_FLAG\\_NEW\\_IMAGE](#)
- [GrOffScreen1BPPSize](#)(i32Width, i32Height)
- [GrOffScreen4BPPSize](#)(i32Width, i32Height)
- [GrOffScreen8BPPSize](#)(i32Width, i32Height)
- [GrPixelDraw](#)(psContext, i32X, i32Y)
- [GrRectContainsPoint](#)(psRect, i32X, i32Y)
- [GrStringBaselineGet](#)(psContext)
- [GrStringDrawCentered](#)(psContext, pcString, i32Length, i32X, i32Y, bOpaque)
- [GrStringHeightGet](#)(psContext)
- [GrStringMaxWidthGet](#)(psContext)
- [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, int32\_t i32X, int32\_t i32Y, int32\_t i32Radius)
- void [GrCircleFill](#) (const [tContext](#) \*pContext, int32\_t i32X, int32\_t i32Y, int32\_t i32Radius)
- void [GrCodepageMapTableSet](#) ([tContext](#) \*psContext, [tCodePointMap](#) \*pCodePointMapTable, uint8\_t ui8NumMaps)
- void [GrContextClipRegionSet](#) ([tContext](#) \*psContext, [tRectangle](#) \*pRect)

- void [GrContextFontSet](#) (tContext \*psContext, const tFont \*pFnt)
- void [GrContextInit](#) (tContext \*psContext, const tDisplay \*psDisplay)
- void [GrDefaultStringRenderer](#) (const tContext \*psContext, const char \*pcString, int32\_t i32Length, int32\_t i32X, int32\_t i32Y, bool bOpaque)
- uint32\_t [GrFontBaselineGet](#) (const tFont \*psFont)
- uint32\_t [GrFontBlockCodepointsGet](#) (const tFont \*psFont, uint16\_t ui16BlockIndex, uint32\_t \*pui32Start)
- uint16\_t [GrFontCodepageGet](#) (const tFont \*psFont)
- const uint8\_t \* [GrFontGlyphDataGet](#) (const tFont \*psFont, uint32\_t ui32CodePoint, uint8\_t \*pui8Width)
- void [GrFontGlyphRender](#) (const tContext \*psContext, const uint8\_t \*pui8Data, int32\_t i32X, int32\_t i32Y, bool bCompressed, bool bOpaque)
- uint32\_t [GrFontHeightGet](#) (const tFont \*psFont)
- void [GrFontInfoGet](#) (const tFont \*psFont, uint8\_t \*pui8Format, uint8\_t \*pui8MaxWidth, uint8\_t \*pui8Height, uint8\_t \*pui8Baseline)
- uint32\_t [GrFontMaxWidthGet](#) (const tFont \*psFont)
- uint16\_t [GrFontNumBlocksGet](#) (const tFont \*psFont)
- void [GrImageDraw](#) (const tContext \*psContext, const uint8\_t \*pui8Image, int32\_t i32X, int32\_t i32Y)
- void [GrLibInit](#) (const tGrLibDefaults \*pDefaults)
- void [GrLineDraw](#) (const tContext \*psContext, int32\_t i32X1, int32\_t i32Y1, int32\_t i32X2, int32\_t i32Y2)
- void [GrLineDrawH](#) (const tContext \*psContext, int32\_t i32X1, int32\_t i32X2, int32\_t i32Y)
- void [GrLineDrawV](#) (const tContext \*psContext, int32\_t i32X, int32\_t i32Y1, int32\_t i32Y2)
- uint32\_t [GrMapISO8859\\_10\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_11\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_13\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_14\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_15\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_16\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_1\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_2\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_3\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_4\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_5\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)
- uint32\_t [GrMapISO8859\\_6\\_Unicode](#) (const char \*pcSrcChar, uint32\_t ui32Count, uint32\_t \*pui32Skip)

- `uint32_t GrMapISO8859_7_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapISO8859_8_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapISO8859_9_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapUnicode_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapUTF16BE_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapUTF16LE_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapUTF8_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapWIN1250_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapWIN1251_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapWIN1252_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapWIN1253_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `uint32_t GrMapWIN1254_Unicode` (`const char *pcSrcChar`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `void GrOffScreen1BPPInit` (`tDisplay *psDisplay`, `uint8_t *pui8Image`, `int32_t i32Width`, `int32_t i32Height`)
- `void GrOffScreen4BPPInit` (`tDisplay *psDisplay`, `uint8_t *pui8Image`, `int32_t i32Width`, `int32_t i32Height`)
- `void GrOffScreen4BPPPaletteSet` (`tDisplay *psDisplay`, `uint32_t *pui32Palette`, `uint32_t ui32Offset`, `uint32_t ui32Count`)
- `void GrOffScreen8BPPInit` (`tDisplay *psDisplay`, `uint8_t *pui8Image`, `int32_t i32Width`, `int32_t i32Height`)
- `void GrOffScreen8BPPPaletteSet` (`tDisplay *psDisplay`, `uint32_t *pui32Palette`, `uint32_t ui32Offset`, `uint32_t ui32Count`)
- `void GrRectDraw` (`const tContext *psContext`, `const tRectangle *psRect`)
- `void GrRectFill` (`const tContext *psContext`, `const tRectangle *psRect`)
- `int32_t GrRectIntersectGet` (`tRectangle *psRect1`, `tRectangle *psRect2`, `tRectangle *psIntersect`)
- `int32_t GrRectOverlapCheck` (`tRectangle *psRect1`, `tRectangle *psRect2`)
- `int32_t GrStringCodepageSet` (`tContext *psContext`, `uint16_t ui16Codepage`)
- `void GrStringDraw` (`const tContext *psContext`, `const char *pcString`, `int32_t i32Length`, `int32_t i32X`, `int32_t i32Y`, `uint32_t bOpaque`)
- `uint32_t GrStringGet` (`int32_t i32Index`, `char *pcData`, `uint32_t ui32Size`)
- `uint32_t GrStringLanguageSet` (`uint16_t ui16LangID`)
- `uint32_t GrStringNextCharGet` (`const tContext *psContext`, `const char *pcString`, `uint32_t ui32Count`, `uint32_t *pui32Skip`)
- `void GrStringTableSet` (`const void *pvTable`)
- `int32_t GrStringWidthGet` (`const tContext *psContext`, `const char *pcString`, `int32_t i32Length`)
- `void GrTransparentImageDraw` (`const tContext *psContext`, `const uint8_t *pui8Image`, `int32_t i32X`, `int32_t i32Y`, `uint32_t ui32Transparent`)

## 3.3.1 Data Structure Documentation

### 3.3.1.1 tCodePointMap

**Definition:**

```
typedef struct
{
    uint16_t ui16SrcCodepage;
    uint16_t ui16FontCodepage;
    uint32_t (*pfnMapChar)(const char *pcSrcChar,
                          uint32_t ui32Count,
                          uint32_t *pui32Skip);
}
tCodePointMap
```

**Members:**

**ui16SrcCodepage** The codepage used to describe the source characters.

**ui16FontCodepage** The codepage into which source characters are to be mapped.

**pfnMapChar** A pointer to the conversion function which will be used to translate input strings into codepoints in the output codepage.

**Description:**

A structure used to define a mapping function that converts text in one codepage to a different codepage. Typically this is used to translate text strings into the codepoints needed to retrieve the appropriate glyphs from a font.

### 3.3.1.2 tContext

**Definition:**

```
typedef struct
{
    int32_t i32Size;
    const tDisplay *psDisplay;
    tRectangle sClipRegion;
    uint32_t ui32Foreground;
    uint32_t ui32Background;
    const tFont *psFont;
    void (*pfnStringRenderer)(const struct _tContext *,
                              const char *,
                              int32_t,
                              int32_t,
                              int32_t,
                              bool);
    const tCodePointMap *pCodePointMapTable;
    uint16_t ui16Codepage;
    uint8_t ui8NumCodePointMaps;
    uint8_t ui8CodePointMap;
    uint8_t ui8Reserved;
}
tContext
```

**Members:**

***i32Size*** The size of this structure.

***psDisplay*** The screen onto which drawing operations are performed.

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

***ui32Foreground*** The color used to draw primitives onto the screen.

***ui32Background*** The background color used to draw primitives onto the screen.

***psFont*** The font used to render text onto the screen.

***pfnStringRenderer*** A pointer to a replacement string rendering function. Applications can use this for language-specific string rendering support. If set, this function is passed control whenever GrStringDraw is called.

***pCodePointMapTable*** A table of functions used to map between the various supported source codepages and the codepages supported by fonts in use.

***ui16Codepage*** The currently selected source text codepage.

***ui8NumCodePointMaps*** The number of entries in the pCodePointMapTable array.

***ui8CodePointMap*** The index of the codepoint map table entry which is currently in use based on the selected source codepage and the current font.

***ui8Reserved*** Reserved for future expansion.

**Description:**

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

### 3.3.1.3 tDisplay

**Definition:**

```
typedef struct
{
    int32_t i32Size;
    void *pvDisplayData;
    uint16_t ui16Width;
    uint16_t ui16Height;
    void (*pfnPixelDraw)(void *pvDisplayData,
                        int32_t i32X,
                        int32_t i32Y,
                        uint32_t ui32Value);
    void (*pfnPixelDrawMultiple)(void *pvDisplayData,
                                int32_t i32X,
                                int32_t i32Y,
                                int32_t i32X0,
                                int32_t i32Count,
                                int32_t i32BPP,
                                const uint8_t *pui8Data,
                                const uint8_t *pui8Palette);
    void (*pfnLineDrawH)(void *pvDisplayData,
                        int32_t i32X1,
                        int32_t i32X2,
                        int32_t i32Y,
                        uint32_t ui32Value);
    void (*pfnLineDrawV)(void *pvDisplayData,
                        int32_t i32X,
```



```

        int32_t i32Y1,
        int32_t i32Y2,
        uint32_t ui32Value);
void (*pfnRectFill)(void *pvDisplayData,
                   const tRectangle *psRect,
                   uint32_t ui32Value);
uint32_t (*pfnColorTranslate)(void *pvDisplayData,
                              uint32_t ui32Value);
void (*pfnFlush)(void *pvDisplayData);
}
tDisplay

```

**Members:**

***i32Size*** The size of this structure.

***pvDisplayData*** A pointer to display driver-specific data.

***ui16Width*** The width of this display.

***ui16Height*** 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. Note that the IBPP parameter contains the source image data color depth in the least significant byte but uses some high bits to pass flags and hints to the driver.

***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.

## 3.3.1.4 tFont

**Definition:**

```

typedef struct
{
    uint8_t ui8Format;
    uint8_t ui8MaxWidth;
    uint8_t ui8Height;
    uint8_t ui8Baseline;
    uint16_t pui16Offset[96];
    const uint8_t *pui8Data;
}
tFont

```

**Members:**

***ui8Format*** The format of the font. Can be one of FONT\_FMT\_UNCOMPRESSED or FONT\_FMT\_PIXEL\_RLE.

***ui8MaxWidth*** 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.

**ui8Height** The height of the character cell; this may be taller than the font data for the characters (to provide inter-line spacing).

**ui8Baseline** 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.

**pui16Offset** The offset within `pui8Data` to the data for each character in the font.

**pui8Data** A pointer to the data for the font.

**Description:**

This structure describes a font used for drawing text onto the screen. Fonts in this format may encode ASCII characters with codepoints in the range 0x20 - 0x7F. More information on this and the other supported font structures may be found in the “Font Format” section of the user’s guide.

### 3.3.1.5 tFontAccessFuncs

**Definition:**

```
typedef struct
{
    void (*pfnFontInfoGet) (uint8_t *pui8FontId,
                           uint8_t *pui8Format,
                           uint8_t *pui8Width,
                           uint8_t *pui8Height,
                           uint8_t *pui8Baseline);
    const uint8_t * (*pfnFontGlyphDataGet) (uint8_t *pui8FontId,
                                           uint32_t ui32CodePoint,
                                           uint8_t *pui8Width);
    uint16_t (*pfnFontCodepageGet) (uint8_t *pui8FontId);
    uint16_t (*pfnFontNumBlocksGet) (uint8_t *pui8FontId);
    uint32_t (*pfnFontBlockCodepointsGet) (uint8_t *pui8FontId,
                                           uint16_t ui16BlockIndex,
                                           uint32_t *pui32Start);
}
tFontAccessFuncs
```

**Members:**

**pfnFontInfoGet** A pointer to the function which will return information on the font. This is used to support `GrFontInfoGet`.

**pfnFontGlyphDataGet** A pointer to the function used to retrieve data for a particular font glyph. This function returns a pointer to the glyph data in linear, random access memory. If a buffer is required to ensure this, that buffer must be owned and managed by the font wrapper function. It is safe to assume that this function will not be called again until any previously requested glyph data has been used so a single character buffer should suffice. This is used to support `GrFontGlyphDataGet`.

**pfnFontCodepageGet** A pointer to the function used to determine the codepage supported by the font.

**pfnFontNumBlocksGet** A pointer to the function used to determine the number of blocks of codepoints supported by the font.

**pfnFontBlockCodepointsGet** A pointer to the function used to determine the codepoints in a given codepoints in a given font block.

**Description:**

The jump table used to access a particular wrapped (offline) font. This table exists for each type of wrapped font in use with the functions dependent upon the storage medium holding the font.

## 3.3.1.6 tFontEx

**Definition:**

```
typedef struct
{
    uint8_t ui8Format;
    uint8_t ui8MaxWidth;
    uint8_t ui8Height;
    uint8_t ui8Baseline;
    uint8_t ui8First;
    uint8_t ui8Last;
    const uint16_t *pui16Offset;
    const uint8_t *pui8Data;
}
tFontEx
```

**Members:**

**ui8Format** The format of the font. Can be one of FONT\_FMT\_EX\_UNCOMPRESSED or FONT\_FMT\_EX\_PIXEL\_RLE.

**ui8MaxWidth** 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.

**ui8Height** The height of the character cell; this may be taller than the font data for the characters (to provide inter-line spacing).

**ui8Baseline** 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.

**ui8First** The codepoint number representing the first character encoded in the font.

**ui8Last** The codepoint number representing the last character encoded in the font.

**pui16Offset** A pointer to a table containing the offset within pui8Data to the data for each character in the font.

**pui8Data** 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. More information on this and the other supported font structures may be found in the “Font Format” section of the user’s guide.

### 3.3.1.7 tFontWide

**Definition:**

```
typedef struct
{
    uint8_t ui8Format;
    uint8_t ui8MaxWidth;
    uint8_t ui8Height;
    uint8_t ui8Baseline;
    uint16_t ui16Codepage;
    uint16_t ui16NumBlocks;
}
tFontWide
```

**Members:**

**ui8Format** The format of the font. Can be one of FONT\_FMT\_WIDE\_UNCOMPRESSED or FONT\_FMT\_WIDE\_PIXEL\_RLE.

**ui8MaxWidth** 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.

**ui8Height** The height of the character cell; this may be taller than the font data for the characters (to provide inter-line spacing).

**ui8Baseline** 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.

**ui16Codepage** The codepage that is used to find characters in this font. This defines the codepoint-to-glyph mapping within this font.

**ui16NumBlocks** The number of blocks of characters described by this font where a block contains a number of contiguous codepoints.

**Description:**

This variant of the font structure supports Unicode and other multi-byte character sets. It is intended for use when rendering such languages as traditional and simplified Chinese, Korean and Japanese. The font supports multiple blocks of contiguous characters and includes a codepage identifier to allow GrLib to correctly map source codepoints to font glyphs in cases where the codepages may differ. More information on this and the other supported font structures may be found in the “Font Format” section of the user’s guide.

### 3.3.1.8 tFontWrapper

**Definition:**

```
typedef struct
{
    uint8_t ui8Format;
    uint8_t *pui8FontId;
    const tFontAccessFuncs *pFuncs;
}
tFontWrapper
```

**Members:**

**ui8Format** The format of the font. Will be FONT\_FMT\_WRAPPED.

***pui8FontId*** A pointer to information required to allow the font access functions to find the font to be used. This value is returned from a call to the FontLoad function for the particular font wrapper in use.

***pFuncs*** Access functions for this font.

**Description:**

This is a wrapper used to support fonts which are stored in a file system or other non-random access storage. The font is accessed by means of access functions whose pointers are described in this structure. The *pui8FontId* field is written with a handle supplied to the application by the font wrapper's FontLoad function and is passed to all access functions to identify the font in use. Wrapped fonts may be used by any GrLib function that accepts a font pointer as a parameter merely by casting the pointer appropriately.

### 3.3.1.9 tGrLibDefaults

**Definition:**

```
typedef struct
{
    void (*pfnStringRenderer)(const struct _tContext *,
                              const char *,
                              int32_t,
                              int32_t,
                              int32_t,
                              bool);

    tCodePointMap *pCodePointMapTable;
    uint16_t ui16Codepage;
    uint8_t ui8NumCodePointMaps;
    uint8_t ui8Reserved;
}
tGrLibDefaults
```

**Members:**

***pfnStringRenderer*** The default string rendering function to use. This will normally be GrDefaultStringRenderer but may be replaced when supporting languages requiring mixed rendering directions such as Arabic or Hebrew.

***pCodePointMapTable*** The default codepoint mapping function table. This table contains information allowing GrLib to map text in the source codepage to the correct glyphs in the fonts to be used. The field points to the first element of an array of *ui8NumCodePointMaps* structures.

***ui16Codepage*** The default source text codepage encoding in use by the application.

***ui8NumCodePointMaps*** The number of entries in the *pCodePointMapTable* array.

***ui8Reserved*** Reserved for future expansion.

**Description:**

This structure contains default values that are set in any new context initialized via a call to GrContextInit. This structure is passed to the graphics library using the GrLibInit function.

### 3.3.1.10 tRectangle

**Definition:**

```
typedef struct
{
    int16_t i16XMin;
    int16_t i16YMin;
    int16_t i16XMax;
    int16_t i16YMax;
}
tRectangle
```

**Members:**

***i16XMin*** The minimum X coordinate of the rectangle.  
***i16YMin*** The minimum Y coordinate of the rectangle.  
***i16XMax*** The maximum X coordinate of the rectangle.  
***i16YMax*** 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.

## 3.3.2 Define Documentation

### 3.3.2.1 CODEPAGE\_ISO8859\_1

**Definition:**

```
#define CODEPAGE_ISO8859_1
```

**Description:**

Identifiers for codepages and source text encoding formats.

### 3.3.2.2 DpyColorTranslate

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

**Definition:**

```
#define DpyColorTranslate(psDisplay,
                          ui32Value)
```

**Parameters:**

***psDisplay*** is the pointer to the display driver structure for the display to operate upon.  
***ui32Value*** 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.

### 3.3.2.3 DpyFlush

Flushes cached drawing operations.

**Definition:**

```
#define DpyFlush(psDisplay)
```

**Parameters:**

***psDisplay*** 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.

### 3.3.2.4 DpyHeightGet

Gets the height of the display.

**Definition:**

```
#define DpyHeightGet(psDisplay)
```

**Parameters:**

***psDisplay*** 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.

### 3.3.2.5 DpyLineDrawH

Draws a horizontal line on a display.

**Definition:**

```
#define DpyLineDrawH(psDisplay,  
                    i32X1,  
                    i32X2,  
                    i32Y,  
                    ui32Value)
```

**Parameters:**

***psDisplay*** is the pointer to the display driver structure for the display to operate upon.

***i32X1*** is the starting X coordinate of the line.

***i32X2*** is the ending X coordinate of the line.

***i32Y*** is the Y coordinate of the line.

***ui32Value*** 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.

### 3.3.2.6 DpyLineDrawV

Draws a vertical line on a display.

**Definition:**

```
#define DpyLineDrawV(psDisplay,  
                    i32X,  
                    i32Y1,  
                    i32Y2,  
                    ui32Value)
```

**Parameters:**

***psDisplay*** is the pointer to the display driver structure for the display to operate upon.

***i32X*** is the X coordinate of the line.

***i32Y1*** is the starting Y coordinate of the line.

***i32Y2*** is the ending Y coordinate of the line.

***ui32Value*** 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.

### 3.3.2.7 DpyPixelDraw

Draws a pixel on a display.

**Definition:**

```
#define DpyPixelDraw(psDisplay,  
                    i32X,  
                    i32Y,  
                    ui32Value)
```

**Parameters:**

***psDisplay*** is the pointer to the display driver structure for the display to operate upon.

***i32X*** is the X coordinate of the pixel.

***i32Y*** is the Y coordinate of the pixel.

***ui32Value*** 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.

### 3.3.2.8 DpyPixelDrawMultiple

Draws a horizontal sequence of pixels on a display.

**Definition:**

```
#define DpyPixelDrawMultiple(psDisplay,  
                             i32X,  
                             i32Y,  
                             i32X0,  
                             i32Count,  
                             i32BPP,  
                             pui8Data,  
                             pui8Palette)
```

**Parameters:**

***psDisplay*** is the pointer to the display driver structure for the display to operate upon.

***i32X*** is the X coordinate of the first pixel.

***i32Y*** is the Y coordinate of the first pixel.

***i32X0*** is sub-pixel offset within the pixel data, which is valid for 1 or 4 bit per pixel formats.

***i32Count*** is the number of pixels to draw.

***i32BPP*** is the number of bits per pixel; must be 1, 4, or 8.

***pui8Data*** 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.

***pui8Palette*** 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.

### 3.3.2.9 DpyRectFill

Fills a rectangle on a display.

**Definition:**

```
#define DpyRectFill(psDisplay,  
                  psRect,  
                  ui32Value)
```

**Parameters:**

***psDisplay*** is the pointer to the display driver structure for the display to operate upon.

***psRect*** is a pointer to the structure describing the rectangle to fill.

**ui32Value** 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.

### 3.3.2.10 DpyWidthGet

Gets the width of the display.

**Definition:**

```
#define DpyWidthGet (psDisplay)
```

**Parameters:**

**psDisplay** 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.

### 3.3.2.11 FONT\_EX\_MARKER

**Definition:**

```
#define FONT_EX_MARKER
```

**Description:**

A marker used in the ui8Format field of a font to indicate that the font data is stored using the new [tFontEx](#) structure.

### 3.3.2.12 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.

### 3.3.2.13 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.

### 3.3.2.14 FONT\_FMT\_PIXEL\_RLE

**Definition:**

```
#define FONT_FMT_PIXEL_RLE
```

**Description:**

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

### 3.3.2.15 FONT\_FMT\_UNCOMPRESSED

**Definition:**

```
#define FONT_FMT_UNCOMPRESSED
```

**Description:**

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

### 3.3.2.16 FONT\_FMT\_WIDE\_PIXEL\_RLE

**Definition:**

```
#define FONT_FMT_WIDE_PIXEL_RLE
```

**Description:**

Indicates that the font data is stored using a pixel-based RLE format and uses the [tFontWide](#) structure format.

### 3.3.2.17 FONT\_FMT\_WIDE\_UNCOMPRESSED

**Definition:**

```
#define FONT_FMT_WIDE_UNCOMPRESSED
```

**Description:**

Indicates that the font data is stored in an uncompressed format and uses the [tFontWide](#) structure format.

### 3.3.2.18 FONT\_FMT\_WRAPPED

**Definition:**

```
#define FONT_FMT_WRAPPED
```

**Description:**

Indicates that the font data is stored in offline storage (file system, serial memory device, etc) and must be accessed using wrapper functions. Fonts using this format are described using a [tFontWrapper](#) structure.

### 3.3.2.19 FONT\_WIDE\_MARKER

**Definition:**

```
#define FONT_WIDE_MARKER
```

**Description:**

A marker used in the `ui8Format` field of a font to indicate that the font data is stored using the new `tFontWide` structure.

### 3.3.2.20 GrContextBackgroundSet

Sets the background color to be used.

**Definition:**

```
#define GrContextBackgroundSet(psContext,  
                               ui32Value)
```

**Parameters:**

***psContext*** is a pointer to the drawing context to modify.  
***ui32Value*** 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.

### 3.3.2.21 GrContextBackgroundSetTranslated

Sets the background color to be used.

**Definition:**

```
#define GrContextBackgroundSetTranslated(psContext,  
                                         ui32Value)
```

**Parameters:**

***psContext*** is a pointer to the drawing context to modify.  
***ui32Value*** 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.

### 3.3.2.22 GrContextDpyHeightGet

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

**Definition:**

```
#define GrContextDpyHeightGet (psContext)
```

**Parameters:**

***psContext*** 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.

### 3.3.2.23 GrContextDpyWidthGet

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

**Definition:**

```
#define GrContextDpyWidthGet (psContext)
```

**Parameters:**

***psContext*** 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.

### 3.3.2.24 GrContextForegroundSet

Sets the foreground color to be used.

**Definition:**

```
#define GrContextForegroundSet (psContext,  
                               ui32Value)
```

**Parameters:**

***psContext*** is a pointer to the drawing context to modify.  
***ui32Value*** 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.

### 3.3.2.25 GrContextForegroundSetTranslated

Sets the foreground color to be used.

**Definition:**

```
#define GrContextForegroundSetTranslated(psContext,  
                                       ui32Value)
```

**Parameters:**

***psContext*** is a pointer to the drawing context to modify.

***ui32Value*** is the display driver-specific color to be used.

**Description:**

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

**Returns:**

None.

### 3.3.2.26 GrFlush

Flushes any cached drawing operations.

**Definition:**

```
#define GrFlush(psContext)
```

**Parameters:**

***psContext*** 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.

### 3.3.2.27 GrImageColorsGet

Gets the number of colors in an image.

**Definition:**

```
#define GrImageColorsGet(pui8Image)
```

**Parameters:**

***pui8Image*** 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.

### 3.3.2.28 GrImageHeightGet

Gets the height of an image.

**Definition:**

```
#define GrImageHeightGet (pui8Image)
```

**Parameters:**

*pui8Image* 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.

### 3.3.2.29 GrImageWidthGet

Gets the width of an image.

**Definition:**

```
#define GrImageWidthGet (pui8Image)
```

**Parameters:**

*pui8Image* 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.

### 3.3.2.30 GRLIB\_DRIVER\_FLAG\_NEW\_IMAGE

**Definition:**

```
#define GRLIB_DRIVER_FLAG_NEW_IMAGE
```

**Description:**

This flag is passed to display driver's PixelDrawMultiple calls in the i32BPP parameter to indicate that a given span of pixels represents the first line of a new image. Drivers may use this to recalculate any color mapping table required to draw the image rather than doing this on every line of pixels.

### 3.3.2.31 GrOffScreen1BPPSize

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

**Definition:**

```
#define GrOffScreen1BPPSize(i32Width,  
                           i32Height)
```

**Parameters:**

***i32Width*** is the width of the image in pixels.

***i32Height*** 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.

### 3.3.2.32 GrOffScreen4BPPSize

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

**Definition:**

```
#define GrOffScreen4BPPSize(i32Width,  
                           i32Height)
```

**Parameters:**

***i32Width*** is the width of the image in pixels.

***i32Height*** 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.

### 3.3.2.33 GrOffScreen8BPPSize

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

**Definition:**

```
#define GrOffScreen8BPPSize(i32Width,  
                           i32Height)
```

**Parameters:**

***i32Width*** is the width of the image in pixels.

***i32Height*** 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.

### 3.3.2.34 GrPixelDraw

Draws a pixel.

**Definition:**

```
#define GrPixelDraw(psContext,  
                  i32X,  
                  i32Y)
```

**Parameters:**

***psContext*** is a pointer to the drawing context to use.

***i32X*** is the X coordinate of the pixel.

***i32Y*** is the Y coordinate of the pixel.

**Description:**

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

**Returns:**

None.

### 3.3.2.35 GrRectContainsPoint

Determines if a point lies within a given rectangle.

**Definition:**

```
#define GrRectContainsPoint(psRect,  
                           i32X,  
                           i32Y)
```

**Parameters:**

***psRect*** is a pointer to the rectangle which the point is to be checked against.

***i32X*** is the X coordinate of the point to be checked.

***i32Y*** is the Y coordinate of the point to be checked.

**Description:**

This function determines whether point (*i32X*, *i32Y*) lies within the rectangle described by *psRect*.

**Returns:**

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

### 3.3.2.36 GrStringBaselineGet

Gets the baseline of a string.

**Definition:**

```
#define GrStringBaselineGet (psContext)
```

**Parameters:**

***psContext*** 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.

### 3.3.2.37 GrStringDrawCentered

Draws a centered string.

**Definition:**

```
#define GrStringDrawCentered (psContext,  
                             pcString,  
                             i32Length,  
                             i32X,  
                             i32Y,  
                             bOpaque)
```

**Parameters:**

***psContext*** is a pointer to the drawing context to use.

***pcString*** is a pointer to the string to be drawn.

***i32Length*** is the number of characters from the string that should be drawn on the screen.

***i32X*** is the X coordinate of the center of the string position on the screen.

***i32Y*** 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 text on the screen centered upon the provided position. The *i32Length* 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.

### 3.3.2.38 GrStringHeightGet

Gets the height of a string.

**Definition:**

```
#define GrStringHeightGet (psContext)
```

**Parameters:**

***psContext*** 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.

### 3.3.2.39 GrStringMaxWidthGet

Gets the maximum width of a character in a string.

**Definition:**

```
#define GrStringMaxWidthGet (psContext)
```

**Parameters:**

***psContext*** 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.

### 3.3.2.40 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.

### 3.3.2.41 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.

### 3.3.2.42 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.

### 3.3.2.43 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.

### 3.3.2.44 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.

### 3.3.2.45 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.

## 3.3.3 Function Documentation

### 3.3.3.1 GrCircleDraw

Draws a circle.

**Prototype:**

```
void  
GrCircleDraw(const tContext *pContext,  
              int32_t i32X,  
              int32_t i32Y,  
              int32_t i32Radius)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to use.  
***i32X*** is the X coordinate of the center of the circle.  
***i32Y*** is the Y coordinate of the center of the circle.  
***i32Radius*** 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  $i32X - i32Radius$  to  $i32X + i32Radius$  and  $i32Y - i32Radius$  to  $i32Y + i32Radius$ , inclusive.

**Returns:**

None.

## 3.3.3.2 GrCircleFill

Draws a filled circle.

**Prototype:**

```
void
GrCircleFill(const tContext *pContext,
             int32_t i32X,
             int32_t i32Y,
             int32_t i32Radius)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to use.  
***i32X*** is the X coordinate of the center of the circle.  
***i32Y*** is the Y coordinate of the center of the circle.  
***i32Radius*** 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  $i32X - i32Radius$  to  $i32X + i32Radius$  and  $i32Y - i32Radius$  to  $i32Y + i32Radius$ , inclusive.

**Returns:**

None.

## 3.3.3.3 GrCodepageMapTableSet

Provides GrLib with a table of source/font codepage mapping functions.

**Prototype:**

```
void
GrCodepageMapTableSet(tContext *pContext,
                     tCodePointMap *pCodePointMapTable,
                     uint8_t ui8NumMaps)
```

**Parameters:**

***pContext*** is a pointer to the context to modify.

***pCodePointMapTable*** points to an array of structures each defining the mapping from a source text codepage to a destination font codepage.

***ui8NumMaps*** provides the number of entries in the *pCodePointMapTable* array.

**Description:**

This function provides GrLib with a set of functions that can be used to map text encoded in a particular codepage to one other codepage. These functions are used to allow GrLib to parse text strings and display the correct glyphs from the font. The mapping function used by the library will depend upon the source text codepage set using a call to [GrStringCodepageSet\(\)](#) and the context's font, set using [GrContextFontSet\(\)](#).

If no conversion function is available to map from the selected source codepage to the font's codepage, GrLib use the first conversion function provided in the codepoint map table and the displayed text will likely be incorrect.

If this call is not made, GrLib assumes ISO8859-1 encoding for both the source text and font to maintain backwards compatibility for applications which were developed prior to the introduction of international character set support.

**Returns:**

None.

### 3.3.3.4 GrContextClipRegionSet

Sets the extents of the clipping region.

**Prototype:**

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

**Parameters:**

***psContext*** 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; *i16XMin* of 1 and *i16XMax* 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.

### 3.3.3.5 GrContextFontSet

Sets the font to be used.

**Prototype:**

```
void  
GrContextFontSet (tContext *pContext,  
                  const tFont *psFont)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to modify.

***psFont*** is a pointer to the font to be used.

**Description:**

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

**Returns:**

None.

### 3.3.3.6 GrContextInit

Initializes a drawing context.

**Prototype:**

```
void  
GrContextInit (tContext *psContext,  
               const tDisplay *psDisplay)
```

**Parameters:**

***psContext*** is a pointer to the drawing context to initialize.

***psDisplay*** 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.

### 3.3.3.7 GrDefaultStringRenderer

The default text string rendering function.

**Prototype:**

```
void  
GrDefaultStringRenderer (const tContext *pContext,  
                          const char *pcString,  
                          int32_t i32Length,  
                          int32_t i32X,  
                          int32_t i32Y,  
                          bool bOpaque)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to use.

***pcString*** is a pointer to the string to be drawn.

***i32Length*** is the number of characters from the string that should be drawn on the screen.

***i32X*** is the X coordinate of the upper left corner of the string position on the screen.

***i32Y*** is the Y coordinate of the upper left corner 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 acts as the default string rendering function called by [GrStringDraw\(\)](#) if no language-specific renderer is registered. It draws a string of text on the screen using the text orientation currently set in the graphics context. The *i32Length* 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).

Applications are not expected to call this function directly but should call [GrStringDraw\(\)](#) instead. This function is provided as an aid to language-specific renders which may call it to render parts of a string at particular positions after dealing with any language-specific layout issues such as, for example, inserting left-to-right numbers within a right-to-left Arabic string.

**Returns:**

None.

### 3.3.3.8 GrFontBaselineGet

Gets the baseline of a font.

**Prototype:**

```
uint32_t  
GrFontBaselineGet (const tFont *psFont)
```

**Parameters:**

***psFont*** 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.

### 3.3.3.9 GrFontBlockCodepointsGet

Returns the number of blocks of character encoded by a font.

**Prototype:**

```
uint32_t  
GrFontBlockCodepointsGet (const tFont *psFont,  
                          uint16_t ui16BlockIndex,  
                          uint32_t *pui32Start)
```



**Parameters:**

***psFont*** is a pointer to the font which is to be queried.

***ui16BlockIndex*** is the index of the codepoint block to be queried.

***pui32Start*** points to storage which is written with the codepoint number of the first glyph in the block.

**Description:**

This function may be used to query the contents of a particular block of codepoints (characters) encoded by a given font. This is primarily of use to applications which wish to parse fonts directly to, for example, display all glyphs in the font. It is unlikely that applications which wish to display text strings would need to call this function.

The number of blocks in the font may be queried by calling [GrFontNumBlocksGet\(\)](#). The *ui16BlockIndex* selects a block and valid values are from, 0 to the number of blocks in the font - 1.

The *pui32Start* pointer is written with the codepoint number of the first glyph in the given block. It is assumed that each block contains a contiguous block of glyphs so the actual codepoints represented in the block will be from *\*pui32Start* to (*\*pui32Start* + return value - 1).

The *psFont* parameter may point to any supported font format including wrapped fonts described using the [tFontWrapper](#) structure (assuming, of course, that the structure pointer is cast to a [tFont](#) pointer).

**Returns:**

Returns the number of blocks of codepoints within the block.

### 3.3.3.10 GrFontCodepageGet

Returns the codepage supported by the given font.

**Prototype:**

```
uint16_t
GrFontCodepageGet (const tFont *psFont)
```

**Parameters:**

***psFont*** points to the font whose codepage is to be returned.

**Description:**

This function returns the codepage supported by the font whose pointer is passed. The codepage defines the mapping between a given character code and the glyph that represents it. Standard codepages are identified by labels of the form **CODEPAGE\_xxxx**. Fonts may also be encoded using application specific codepages with values of 0x8000 or higher.

**Returns:**

Returns the font codepage identifier.

### 3.3.3.11 GrFontGlyphDataGet

Retrieves a pointer to the data for a specific font glyph.

**Prototype:**

```
const uint8_t *
GrFontGlyphDataGet(const tFont *psFont,
                   uint32_t ui32CodePoint,
                   uint8_t *pui8Width)
```

**Parameters:**

**psFont** points to the font whose glyph is to be queried.

**ui32CodePoint** identifies the specific glyph whose data is being queried.

**pui8Width** points to storage which will be written with the width of the requested glyph in pixels.

**Description:**

This function may be used to retrieve the pixel data for a particular glyph in a font. The pointer returned may be passed to GrFontGlyphRender to draw the glyph on the display. The format of the data may be determined from the font format returned via a call to [GrFontInfoGet\(\)](#).

**Returns:**

Returns a pointer to the data for the requested glyph or NULL if the glyph does not exist in the font.

### 3.3.3.12 GrFontGlyphRender

Renders a single character glyph on the display at a given position.

**Prototype:**

```
void
GrFontGlyphRender(const tContext *pContext,
                  const uint8_t *pui8Data,
                  int32_t i32X,
                  int32_t i32Y,
                  bool bCompressed,
                  bool bOpaque)
```

**Parameters:**

**pContext** points to the graphics context in use.

**pui8Data** points to the first byte of data for the glyph to be rendered.

**i32X** is the X coordinate of the top left pixel of the glyph.

**i32Y** is the Y coordinate of the top left pixel of the glyph.

**bCompressed** is **true** if the data pointed to by **pui8Data** is in compressed format or **false** if uncompressed.

**bOpaque** is **true** if background pixels are to be written or **false** if only foreground pixels are drawn.

**Description:**

This function is included as an aid to language-specific string rendering functions. Applications are expected to render strings and characters using calls to GrStringDraw or GrStringDraw-Centered and should not call this function directly.

A string rendering function may call this low level API to place a single character glyph on the display at a particular position. The rendered glyph is subject to the clipping rectangle currently set in the passed graphics context. Rendering colors are also taken from the context structure. Glyph data pointed to by **pui8Data** should be retrieved using a call to [GrFontGlyphDataGet\(\)](#).

**Returns:**  
None.

### 3.3.3.13 GrFontHeightGet

Gets the height of a font.

**Prototype:**

```
uint32_t  
GrFontHeightGet (const tFont *psFont)
```

**Parameters:**

***psFont*** 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.

### 3.3.3.14 GrFontInfoGet

Retrieves header information from a font.

**Prototype:**

```
void  
GrFontInfoGet (const tFont *psFont,  
               uint8_t *pui8Format,  
               uint8_t *pui8MaxWidth,  
               uint8_t *pui8Height,  
               uint8_t *pui8Baseline)
```

**Parameters:**

***psFont*** points to the font whose information is to be queried.

***pui8Format*** points to storage which will be written with the font format.

***pui8MaxWidth*** points to storage which will be written with the maximum character width for the font in pixels.

***pui8Height*** points to storage which will be written with the height of the font character cell in pixels.

***pui8Baseline*** points to storage which will be written with the font baseline offset in pixels.

**Description:**

This function may be used to retrieve information about a given font. The *psFont* parameter may point to any supported font format including wrapped fonts described using a [tFontWrapper](#) structure (with the pointer cast to a [tFont](#) pointer).

**Returns:**

None.

### 3.3.3.15 GrFontMaxWidthGet

Gets the maximum width of a font.

**Prototype:**

```
uint32_t  
GrFontMaxWidthGet (const tFont *psFont)
```

**Parameters:**

*psFont* 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.

### 3.3.3.16 GrFontNumBlocksGet

Returns the number of blocks of character encoded by a font.

**Prototype:**

```
uint16_t  
GrFontNumBlocksGet (const tFont *psFont)
```

**Parameters:**

*psFont* is a pointer to the font which is to be queried.

**Description:**

This function may be used to query the number of contiguous blocks of codepoints (characters) encoded by a given font. This is primarily of use to applications which wish to parse fonts directly to, for example, display all glyphs in the font. It is unlikely that applications which wish to display text strings would need to call this function.

The *psFont* parameter may point to any supported font format including wrapped fonts described using the *tFontWrapper* structure (assuming, of course, that the structure pointer is cast to a *tFont* pointer).

**Returns:**

Returns the number of blocks of codepoints within the font.

### 3.3.3.17 GrImageDraw

Draws a bitmap image.

**Prototype:**

```
void  
GrImageDraw (const tContext *pContext,  
             const uint8_t *pui8Image,  
             int32_t i32X,  
             int32_t i32Y)
```

**Parameters:**

*pContext* is a pointer to the drawing context to use.

*pui8Image* is a pointer to the image to draw.

*i32X* is the X coordinate of the upper left corner of the image.

*i32Y* 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.

### 3.3.3.18 GrLibInit

Initializes graphics library default text rendering values.

**Prototype:**

```
void  
GrLibInit(const tGrLibDefaults *pDefaults)
```

**Parameters:**

*pDefaults* points to a structure containing default values to use.

**Description:**

This function allows an application to set global default values that the graphics library will use when initializing any graphics context. These values set the source text codepage, the rendering function to use for strings and mapping functions used to allow extraction of the correct glyphs from fonts.

If this function is not called by an application, the graphics library assumes that text strings are ISO8859-1 encoded and that the default string renderer is used.

**Returns:**

None.

### 3.3.3.19 GrLineDraw

Draws a line.

**Prototype:**

```
void  
GrLineDraw(const tContext *pContext,  
            int32_t i32X1,  
            int32_t i32Y1,  
            int32_t i32X2,  
            int32_t i32Y2)
```

**Parameters:**

*pContext* is a pointer to the drawing context to use.

*i32X1* is the X coordinate of the start of the line.

*i32Y1* is the Y coordinate of the start of the line.

*i32X2* is the X coordinate of the end of the line.

*i32Y2* 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 clipping rectangle using the Cohen-Sutherland clipping algorithm, and then scan converted using Bresenham's line drawing algorithm.

**Returns:**

None.

### 3.3.3.20 GrLineDrawH

Draws a horizontal line.

**Prototype:**

```
void
GrLineDrawH(const tContext *pContext,
             int32_t i32X1,
             int32_t i32X2,
             int32_t i32Y)
```

**Parameters:**

*pContext* is a pointer to the drawing context to use.

*i32X1* is the X coordinate of one end of the line.

*i32X2* is the X coordinate of the other end of the line.

*i32Y* 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.

### 3.3.3.21 GrLineDrawV

Draws a vertical line.

**Prototype:**

```
void
GrLineDrawV(const tContext *pContext,
             int32_t i32X,
             int32_t i32Y1,
             int32_t i32Y2)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to use.  
***i32X*** is the X coordinate of the line.  
***i32Y1*** is the Y coordinate of one end of the line.  
***i32Y2*** 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.

### 3.3.3.22 GrMapISO8859\_10\_Unicode

Maps an ISO8859-10 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_10_Unicode(const char *pcSrcChar,
                        uint32_t ui32Count,
                        uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing ISO8859-10 encoded text.  
***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.  
***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-10 format into 32 bit Unicode typically used by wide character fonts. Character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF are converted using a global data table.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-10.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

### 3.3.3.23 GrMapISO8859\_11\_Unicode

Maps an ISO8859-11 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_11_Unicode(const char *pcSrcChar,
                        uint32_t ui32Count,
                        uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing ISO8859-11 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-11 format into 32 bit Unicode typically used by wide character fonts. This conversion is straightforward since character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF map to the Unicode by adding 0xD60 to the ISO8859-11 code.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-11.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

### 3.3.3.24 GrMapISO8859\_13\_Unicode

Maps an ISO8859-13 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapISO8859_13_Unicode(const char *pcSrcChar,  
                        uint32_t ui32Count,  
                        uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing ISO8859-13 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-13 format into 32 bit Unicode typically used by wide character fonts. Character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF are converted using a global data table.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-13.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

### 3.3.3.25 GrMapISO8859\_14\_Unicode

Maps an ISO8859-14 encoded character to its Unicode equivalent.



**Prototype:**

```
uint32_t
GrMapISO8859_14_Unicode(const char *pcSrcChar,
                        uint32_t ui32Count,
                        uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-14 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-9 format into 32 bit Unicode typically used by wide character fonts. This character set is similar to ISO8859-1 but substitutes several characters to offer support for various Celtic languages.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-14.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.26 GrMapISO8859\_15\_Unicode

Maps an ISO8859-15 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_15_Unicode(const char *pcSrcChar,
                        uint32_t ui32Count,
                        uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-15 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-15 format into 32 bit Unicode typically used by wide character fonts. This character set is similar to ISO8859-1 differing from that codepage in only 8 positions.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-15.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.27 GrMapISO8859\_16\_Unicode

Maps an ISO8859-16 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_16_Unicode(const char *pcSrcChar,
                        uint32_t ui32Count,
                        uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-16 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-16 format into 32 bit Unicode typically used by wide character fonts. Character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF are converted using a global data table.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-16.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.28 GrMapISO8859\_1\_Unicode

Maps an ISO8859-1 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_1_Unicode(const char *pcSrcChar,
                       uint32_t ui32Count,
                       uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-1 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-1 format into 32 bit Unicode typically used by wide character fonts. This conversion is trivial since the bottom 256 characters in Unicode are the ISO8859-1 characters and since ISO8859-1 is not a variable length encoding (every character is exactly 1 byte).

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-1.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the `pcSrcChar` string passed.

### 3.3.3.29 GrMapISO8859\_2\_Unicode

Maps an ISO8859-2 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapISO8859_2_Unicode(const char *pcSrcChar,  
                       uint32_t ui32Count,  
                       uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing ISO8859-2 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by `pcSrcChar`.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in `pcSrcChar` to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-2 format into 32 bit Unicode typically used by wide character fonts. This conversion is straightforward since character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF are converted using a global data table.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-2.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the `pcSrcChar` string passed.

### 3.3.3.30 GrMapISO8859\_3\_Unicode

Maps an ISO8859-3 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapISO8859_3_Unicode(const char *pcSrcChar,  
                       uint32_t ui32Count,  
                       uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing ISO8859-3 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by `pcSrcChar`.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in `pcSrcChar` to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-3 format into 32 bit Unicode typically used by wide character fonts.

This conversion is straightforward since character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF are converted using a global data table.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-3.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.31 GrMapISO8859\_4\_Unicode

Maps an ISO8859-4 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_4_Unicode(const char *pcSrcChar,
                       uint32_t ui32Count,
                       uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-4 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-4 format into 32 bit Unicode typically used by wide character fonts. This conversion is straightforward since character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF are converted using a global data table.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-4.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.32 GrMapISO8859\_5\_Unicode

Maps an ISO8859-5 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_5_Unicode(const char *pcSrcChar,
                       uint32_t ui32Count,
                       uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-5 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in `pcSrcChar` to get to the next character in the buffer.

**Description:**

This function may be passed to `GrCodepageMapTableSet()` in a `tCodePointMap` structure to map source text in ISO8859-5 format into 32 bit Unicode typically used by wide character fonts. This conversion is straightforward since character codes 0xA0 and below map directly to the same code in Unicode and those from 0xA1 to 0xFF map to the Unicode by adding 0x360 to the ISO8859-5 code.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-5.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the `pcSrcChar` string passed.

### 3.3.3.33 GrMapISO8859\_6\_Unicode

Maps an ISO8859-6 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_6_Unicode(const char *pcSrcChar,
                       uint32_t ui32Count,
                       uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing ISO8859-6 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by `pcSrcChar`.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in `pcSrcChar` to get to the next character in the buffer.

**Description:**

This function may be passed to `GrCodepageMapTableSet()` in a `tCodePointMap` structure to map source text in ISO8859-6 format into 32 bit Unicode typically used by wide character fonts. This conversion is straightforward since character codes 0xA4 and below map directly to the same code in Unicode and those from 0xA5 to 0xFF map to the Unicode by adding 0x560 to the ISO8859-5 code. The only odd-man-out is character 0xAD which maps to Unicode 0xAD even though it is above 0xA4.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-6.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the `pcSrcChar` string passed.

### 3.3.3.34 GrMapISO8859\_7\_Unicode

Maps an ISO8859-7 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapISO8859_7_Unicode(const char *pcSrcChar,  
                      uint32_t ui32Count,  
                      uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-7 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-7 format into 32 bit Unicode typically used by wide character fonts. This conversion is straightforward since character codes 0xA0 and below map directly to the same code in Unicode, those from 0xBE to 0xFE map to the Unicode by adding 0x2D0 to the ISO8859-7 code and only a few "unusual" mappings exist between codepoints 0xA1 and 0xBD.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-7.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.35 GrMapISO8859\_8\_Unicode

Maps an ISO8859-8 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapISO8859_8_Unicode(const char *pcSrcChar,  
                      uint32_t ui32Count,  
                      uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-8 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-8 format into 32 bit Unicode typically used by wide character fonts. This conversion is straightforward since character codes 0xDE and below map directly to the same code in Unicode, those from 0xE0 to 0xFA map to the Unicode by adding 0x4F0 to the ISO8859-7 code and only a few "unusual" mappings exist at codepoints 0xDF, 0xFD and 0xFE.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-8.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.36 GrMapISO8859\_9\_Unicode

Maps an ISO8859-9 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapISO8859_9_Unicode(const char *pcSrcChar,
                       uint32_t ui32Count,
                       uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing ISO8859-9 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in ISO8859-9 format into 32 bit Unicode typically used by wide character fonts. This character set is almost identical to ISO8859-1 but substitutes 6 characters to offer better support for the Turkish language.

See <http://unicode.org/Public/MAPPINGS/ISO8859/8859-9.TXT> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.37 GrMapUnicode\_Unicode

Maps an 32 bit Unicode encoded character to itself.

**Prototype:**

```
uint32_t
GrMapUnicode_Unicode(const char *pcSrcChar,
                     uint32_t ui32Count,
                     uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing 32 bit Unicode text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in Unicode (UTF32) format into 32 bit Unicode typically used by wide character fonts. This identity conversion is trivial - we merely read 4 bytes at a time and return the 32 bit value they contain. It is assumed that the text is encoded in little endian format.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.38 GrMapUTF16BE\_Unicode

Maps a UTF-16BE encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapUTF16BE_Unicode(const char *pcSrcChar,  
                    uint32_t ui32Count,  
                    uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing UTF-16BE encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in UTF-16BE format into 32 bit Unicode typically used by wide character fonts. This conversion will read bytes from the buffer and decode the first full UTF-16BE character found, returning the Unicode code for that character and the number of bytes to advance *pcSrcChar* by to point to the end of the decoded character. If no valid UTF-16BE character is found, 0 is returned.

See <http://en.wikipedia.org/wiki/UTF-16> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

### 3.3.3.39 GrMapUTF16LE\_Unicode

Maps a UTF-16LE encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapUTF16LE_Unicode(const char *pcSrcChar,  
                    uint32_t ui32Count,  
                    uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing UTF-16LE encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in UTF-16LE format into 32 bit Unicode typically used by wide character fonts. This conversion will read bytes from the buffer and decode the first full UTF-16LE character found, returning the Unicode code for that character and the number of bytes to advance *pcSrcChar* by to point to the end of the decoded character. If no valid UTF-16LE character is found, 0 is returned.



See <http://en.wikipedia.org/wiki/UTF-16> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

### 3.3.3.40 GrMapUTF8\_Unicode

Maps a UTF-8 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapUTF8_Unicode(const char *pcSrcChar,  
                  uint32_t ui32Count,  
                  uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing UTF-8 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in UTF-8 format into 32 bit Unicode typically used by wide character fonts. This conversion will read bytes from the buffer and decode the first full UTF-8 character found, returning the Unicode code for that character and the number of bytes to advance *pcSrcChar* by to point to the end of the decoded character. If no valid UTF-8 character is found, 0 is returned.

See <http://en.wikipedia.org/wiki/UTF-8> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

### 3.3.3.41 GrMapWIN1250\_Unicode

Maps a WIN1250 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapWIN1250_Unicode(const char *pcSrcChar,  
                     uint32_t ui32Count,  
                     uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing WIN1250 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in WIN1250 format into 32 bit Unicode typically used by wide character fonts. Windows-1250 is a codepage commonly used in processing eastern European text. This conversion is straightforward since character codes 0x7F and below map directly to the same code in Unicode, and those from 0x80 to 0xFF are converted using a global data table.

See <http://en.wikipedia.org/wiki/Windows-1250> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.42 GrMapWIN1251\_Unicode

Maps a WIN1251 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapWIN1251_Unicode(const char *pcSrcChar,
                    uint32_t ui32Count,
                    uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing WIN1251 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in WIN1251 format into 32 bit Unicode typically used by wide character fonts. Windows-1251 is a codepage commonly used in processing Cyrillic text. This conversion is straightforward since character codes 0x7F and below map directly to the same code in Unicode, those from 0x80 to 0xBF are converted using a global data table and those from 0xC0 to 0xFF map to the Unicode by adding 0x350 to the WIN1251 code.

See <http://en.wikipedia.org/wiki/Windows-1251> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.43 GrMapWIN1252\_Unicode

Maps a WIN1252 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapWIN1252_Unicode(const char *pcSrcChar,
                    uint32_t ui32Count,
                    uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing WIN1252 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in WIN1252 format into 32 bit Unicode typically used by wide character fonts. Windows-1252 is a codepage commonly used in processing western European text. This conversion is straightforward since character codes 0x7F and below, and 0xA0 and above map directly to the same code in Unicode, and those from 0x80 to 0x9F are converted using a global data table. This codepage can be thought of as a superset of ISO8859-1 and text purported to be encoded in ISO8859-1 is frequently processed using this codepage instead.

See <http://en.wikipedia.org/wiki/Windows-1252> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

## 3.3.3.44 GrMapWIN1253\_Unicode

Maps a WIN1253 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t
GrMapWIN1253_Unicode(const char *pcSrcChar,
                    uint32_t ui32Count,
                    uint32_t *pui32Skip)
```

**Parameters:**

***pcSrcChar*** is a pointer to a string containing WIN1253 encoded text.

***ui32Count*** is the number of bytes in the buffer pointed to by *pcSrcChar*.

***pui32Skip*** points to storage that will be written with the number of bytes to skip in *pcSrcChar* to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in WIN1253 format into 32 bit Unicode typically used by wide character fonts. Windows-1253 is a codepage commonly used in processing Greek text. This conversion is straightforward since character codes 0x7F and below map directly to the same code in Unicode, those from 0x80 to 0xBF are converted using a global data table and those from 0xC0 to 0xFF map to the Unicode by adding 0x2D0 to the WIN1253 code.

See <http://en.wikipedia.org/wiki/Windows-1253> for more information.

**Returns:**

Returns the Unicode character code for the first character in the *pcSrcChar* string passed.

## 3.3.3.45 GrMapWIN1254\_Unicode

Maps a WIN1254 encoded character to its Unicode equivalent.

**Prototype:**

```
uint32_t  
GrMapWIN1254_Unicode(const char *pcSrcChar,  
                    uint32_t ui32Count,  
                    uint32_t *pui32Skip)
```

**Parameters:**

**pcSrcChar** is a pointer to a string containing WIN1254 encoded text.

**ui32Count** is the number of bytes in the buffer pointed to by pcSrcChar.

**pui32Skip** points to storage that will be written with the number of bytes to skip in pcSrcChar to get to the next character in the buffer.

**Description:**

This function may be passed to [GrCodepageMapTableSet\(\)](#) in a [tCodePointMap](#) structure to map source text in WIN1254 format into 32 bit Unicode typically used by wide character fonts. Windows-1254 is a codepage commonly used in processing Turkish text. It is compatible with ISO8859-9 but adds several printable characters in the 0x80-0x9F range.

See <http://en.wikipedia.org/wiki/Windows-1254> for more information.

**Returns:**

Returns the Unicode character code for the first character in the pcSrcChar string passed.

### 3.3.3.46 GrOffScreen1BPPInit

Initializes a 1 BPP off-screen buffer.

**Prototype:**

```
void  
GrOffScreen1BPPInit(tDisplay *psDisplay,  
                  uint8_t *pui8Image,  
                  int32_t i32Width,  
                  int32_t i32Height)
```

**Parameters:**

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

**pui8Image** is a pointer to the image buffer to be used for the off-screen buffer.

**i32Width** is the width of the image buffer in pixels.

**i32Height** 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.

### 3.3.3.47 GrOffScreen4BPPInit

Initializes a 4 BPP off-screen buffer.

**Prototype:**

```
void  
GrOffScreen4BPPInit (tDisplay *psDisplay,  
                    uint8_t *pui8Image,  
                    int32_t i32Width,  
                    int32_t i32Height)
```

**Parameters:**

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

**pui8Image** is a pointer to the image buffer to be used for the off-screen buffer.

**i32Width** is the width of the image buffer in pixels.

**i32Height** 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.

### 3.3.3.48 GrOffScreen4BPPPaletteSet

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

**Prototype:**

```
void  
GrOffScreen4BPPPaletteSet (tDisplay *psDisplay,  
                          uint32_t *pui32Palette,  
                          uint32_t ui32Offset,  
                          uint32_t ui32Count)
```

**Parameters:**

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

**pui32Palette** is a pointer to the array of 24-bit RGB values to be placed into the palette.

**ui32Offset** is the starting offset into the image palette.

**ui32Count** 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.

### 3.3.3.49 GrOffScreen8BPPInit

Initializes an 8 BPP off-screen buffer.

**Prototype:**

```
void  
GrOffScreen8BPPInit (tDisplay *psDisplay,  
                    uint8_t *pui8Image,  
                    int32_t i32Width,  
                    int32_t i32Height)
```

**Parameters:**

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

***pui8Image*** is a pointer to the image buffer to be used for the off-screen buffer.

***i32Width*** is the width of the image buffer in pixels.

***i32Height*** 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.

### 3.3.3.50 GrOffScreen8BPPPaletteSet

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

**Prototype:**

```
void  
GrOffScreen8BPPPaletteSet (tDisplay *psDisplay,  
                          uint32_t *pui32Palette,  
                          uint32_t ui32Offset,  
                          uint32_t ui32Count)
```

**Parameters:**

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

***pui32Palette*** is a pointer to the array of 24-bit RGB values to be placed into the palette.

***ui32Offset*** is the starting offset into the image palette.

***ui32Count*** 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.

### 3.3.3.51 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 *i32XMin* to *i32XMax* and *i32YMin* to *i32YMax*, inclusive.

**Returns:**

None.

### 3.3.3.52 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 *i32XMin* to *i32XMax* and *i32YMin* to *i32YMax*, 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.

### 3.3.3.53 GrRectIntersectGet

Determines the intersection of two rectangles.

**Prototype:**

```
int32_t  
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.

### 3.3.3.54 GrRectOverlapCheck

Determines if two rectangles overlap.

**Prototype:**

```
int32_t  
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  $i16XMin < i16XMax$  and  $i16YMin < i16YMax$ .

**Returns:**

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

### 3.3.3.55 GrStringCodepageSet

Sets the source text codepage to be used.

**Prototype:**

```
int32_t  
GrStringCodepageSet (tContext *pContext,  
                   uint16_t ui16Codepage)
```

**Parameters:**

***pContext*** is a pointer to the context to modify.

***ui16Codepage*** is the identifier of the codepage for the text that the application will pass on future calls to the [GrStringDraw\(\)](#) and [GrStringDrawCentered\(\)](#) functions.

**Description:**

This function sets the codepage that will be used when rendering text via future calls to [GrStringDraw\(\)](#) or [GrStringDrawCentered\(\)](#). The codepage defines the mapping between specific numbers used to define characters and the actual character glyphs displayed. By default, GrLib assumes text passed is encoded using the ISO8859-1 which supports ASCII and western



European character sets. Applications wishing to use multi-byte character sets or alphabets other than those supported by ISO8859-1 should set an appropriate codepage such as UTF-8.

It is important to ensure that your application makes use of fonts which support the required codepage or that you have supplied GrLib with a codepage mapping function that allows translation of your chosen text codepage into the codepage supported by the fonts in use. Several mapping functions for commonly-used codepages are provided and others can be written easily to support different text and font codepage combinations. Codepage mapping functions are provided to GrLib in a table passed as a parameter to the function [GrCodepageMapTableSet\(\)](#).

**Returns:**

None.

### 3.3.3.56 GrStringDraw

Draws a string.

**Prototype:**

```
void  
GrStringDraw(const tContext *pContext,  
             const char *pcString,  
             int32_t i32Length,  
             int32_t i32X,  
             int32_t i32Y,  
             uint32_t bOpaque)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to use.

***pcString*** is a pointer to the string to be drawn.

***i32Length*** is the number of characters from the string that should be drawn on the screen.

***i32X*** is the X coordinate of the upper left corner of the string position on the screen.

***i32Y*** is the Y coordinate of the upper left corner 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 text on the screen. The *i32Length* 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.

### 3.3.3.57 GrStringGet

This function returns a string from the current string table.

**Prototype:**

```
uint32_t  
GrStringGet (int32_t i32Index,  
             char *pcData,  
             uint32_t ui32Size)
```

**Parameters:**

***i32Index*** is the index of the string to retrieve.  
***pcData*** is the pointer to the buffer to store the string into.  
***ui32Size*** 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 *iIndex* 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 *ui32Size* parameter.

**Returns:**

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

### 3.3.3.58 GrStringLanguageSet

This function sets the current language for strings returned by the [GrStringGet\(\)](#) function.

**Prototype:**

```
uint32_t  
GrStringLanguageSet (uint16_t ui16LangID)
```

**Parameters:**

***ui16LangID*** 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 *ui16LangID* 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 language was found.

### 3.3.3.59 GrStringNextCharGet

Returns the codepoint of the first character in a string.

**Prototype:**

```
uint32_t  
GrStringNextCharGet (const tContext *pContext,  
                    const char *pcString,  
                    uint32_t ui32Count,  
                    uint32_t *pui32Skip)
```

**Parameters:**

***pContext*** points to the graphics context in use.

***pcString*** points to the first byte of the string from which the next character is to be parsed.

***ui32Count*** provides the number of bytes in the *pcString* buffer.

***pui32Skip*** points to storage which will be written with the number of bytes that must be skipped in the string buffer to move past the current character.

**Description:**

This function is used to walk through a string extracting one character at a time. The input string is assumed to be encoded using the currently- selected string codepage (as set via a call to the [GrStringCodepageSet\(\)](#) function). The value returned is the codepoint of the first character in the string as mapped into the current font's codepage. This may be passed to the [GrFontGlyphDataGet\(\)](#) function to retrieve the glyph data for the character.

Since variable length encoding schemes such as UTF-8 are supported, this function also returns information on the size of the character that has been parsed, allowing the caller to increment the string pointer by the relevant amount before querying the next character in the string.

**Returns:**

Returns the font codepoint representing the first character in the string or 0 if no valid character was found.

## 3.3.3.60 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.

## 3.3.3.61 GrStringWidthGet

Determines the width of a string.

**Prototype:**

```
int32_t
GrStringWidthGet(const tContext *pContext,
                 const char *pcString,
                 int32_t i32Length)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to use.

***pcString*** is the string in question.

***i32Length*** 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 *i32Length* 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.

### 3.3.3.62 GrTransparentImageDraw

Draws a bitmap image, dropping out a single transparent color.

**Prototype:**

```
void  
GrTransparentImageDraw(const tContext *pContext,  
                      const uint8_t *pui8Image,  
                      int32_t i32X,  
                      int32_t i32Y,  
                      uint32_t ui32Transparent)
```

**Parameters:**

***pContext*** is a pointer to the drawing context to use.

***pui8Image*** is a pointer to the image to draw.

***i32X*** is the X coordinate of the upper left corner of the image.

***i32Y*** is the Y coordinate of the upper left corner of the image.

***ui32Transparent*** is the image color which is to be considered transparent.

**Description:**

This function draws a bitmap image but, unlike GrImageDraw, will drop out any pixel of a particular color allowing the previous background to “shine through”. 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). For 4bpp and 8bpp images, the **ui32Transparent** parameter contains the palette index of the colour which is to be considered transparent. For 1bpp images, the **ui32Transparent** parameter should be set to 0 to draw only foreground pixels or 1 to draw only background pixels.

**Returns:**

None.

---

## 4 Widget Framework

Introduction .....	85
Definitions .....	85

### 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 `grrlib/widget.c`, with `grrlib/widget.h` containing the API declarations for use by applications.

### 4.2 Definitions

#### Data Structures

- `tWidget`

#### Defines

- `WIDGET_MSG_KEY_DOWN`
- `WIDGET_MSG_KEY_LEFT`
- `WIDGET_MSG_KEY_RIGHT`
- `WIDGET_MSG_KEY_SELECT`
- `WIDGET_MSG_KEY_UP`
- `WIDGET_MSG_PAINT`
- `WIDGET_MSG_PTR_DOWN`
- `WIDGET_MSG_PTR_MOVE`
- `WIDGET_MSG_PTR_UP`
- `WIDGET_ROOT`
- `WidgetPaint(psWidget)`

## Functions

- void `WidgetAdd` (`tWidget *psParent`, `tWidget *psWidget`)
- `int32_t` `WidgetDefaultMsgProc` (`tWidget *psWidget`, `uint32_t ui32Message`, `uint32_t ui32Param1`, `uint32_t ui32Param2`)
- `int32_t` `WidgetMessageQueueAdd` (`tWidget *psWidget`, `uint32_t ui32Message`, `uint32_t ui32Param1`, `uint32_t ui32Param2`, `bool bPostOrder`, `bool bStopOnSuccess`)
- void `WidgetMessageQueueProcess` (`void`)
- `uint32_t` `WidgetMessageSendPostOrder` (`tWidget *psWidget`, `uint32_t ui32Message`, `uint32_t ui32Param1`, `uint32_t ui32Param2`, `bool bStopOnSuccess`)
- `uint32_t` `WidgetMessageSendPreOrder` (`tWidget *psWidget`, `uint32_t ui32Message`, `uint32_t ui32Param1`, `uint32_t ui32Param2`, `bool bStopOnSuccess`)
- `uint32_t` `WidgetMutexGet` (`uint8_t *pi8Mutex`)
- void `WidgetMutexInit` (`uint8_t *pi8Mutex`)
- void `WidgetMutexPut` (`uint8_t *pi8Mutex`)
- `int32_t` `WidgetPointerMessage` (`uint32_t ui32Message`, `int32_t i32X`, `int32_t i32Y`)
- void `WidgetRemove` (`tWidget *psWidget`)

## 4.2.1 Data Structure Documentation

### 4.2.1.1 tWidget

**Definition:**

```
typedef struct
{
    int32_t i32Size;
    tWidget *psParent;
    tWidget *psNext;
    tWidget *psChild;
    const tDisplay *psDisplay;
    tRectangle sPosition;
    int32_t (*pfnMsgProc) (tWidget *psWidget,
                          uint32_t ui32Message,
                          uint32_t ui32Param1,
                          uint32_t ui32Param2);
}
tWidget
```

**Members:**

***i32Size*** The size of this structure. This will be the size of the full structure, not just the generic widget subset.

***psParent*** A pointer to this widget's parent widget.

***psNext*** A pointer to this widget's first sibling widget.

***psChild*** A pointer to this widget's first child widget.

***psDisplay*** 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\_KEY\_DOWN

**Definition:**

```
#define WIDGET_MSG_KEY_DOWN
```

**Description:**

This message is sent by the application to indicate that there has been a key press or button press meaning “down”. *ui32Param1* by convention is a pointer to the widget that is the intended recipient of the key press. This is controlled by the application.

### 4.2.2.2 WIDGET\_MSG\_KEY\_LEFT

**Definition:**

```
#define WIDGET_MSG_KEY_LEFT
```

**Description:**

This message is sent by the application to indicate that there has been a key press or button press meaning “left”. *ui32Param1* by convention is a pointer to the widget that is the intended recipient of the key press. This is controlled by the application.

### 4.2.2.3 WIDGET\_MSG\_KEY\_RIGHT

**Definition:**

```
#define WIDGET_MSG_KEY_RIGHT
```

**Description:**

This message is sent by the application to indicate that there has been a key press or button press meaning “right”. *ui32Param1* by convention is a pointer to the widget that is the intended recipient of the key press. This is controlled by the application.

### 4.2.2.4 WIDGET\_MSG\_KEY\_SELECT

**Definition:**

```
#define WIDGET_MSG_KEY_SELECT
```

**Description:**

This message is sent by the application to indicate that there has been a key press or button press meaning “select”. *ui32Param1* by convention is a pointer to the widget that is the intended recipient of the key press. This is controlled by the application.

#### 4.2.2.5 WIDGET\_MSG\_KEY\_UP

**Definition:**

```
#define WIDGET_MSG_KEY_UP
```

**Description:**

This message is sent by the application to indicate that there has been a key press or button press meaning "up". *ui32Param1* by convention is a pointer to the widget that is the intended recipient of the key press. This is controlled by the application.

#### 4.2.2.6 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 *ui32Param1* nor *ui32Param2* are used by this message. This message is delivered in top-down order.

#### 4.2.2.7 WIDGET\_MSG\_PTR\_DOWN

**Definition:**

```
#define WIDGET_MSG_PTR_DOWN
```

**Description:**

This message is sent to indicate that the pointer is now down. *ui32Param1* is the X coordinate of the location where the pointer down event occurred, and *ui32Param2* is the Y coordinate. This message is delivered in bottom-up order.

#### 4.2.2.8 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. *ui32Param1* is the X coordinate of the new pointer location, and *ui32Param2* is the Y coordinate. This message is delivered in bottom-up order.

#### 4.2.2.9 WIDGET\_MSG\_PTR\_UP

**Definition:**

```
#define WIDGET_MSG_PTR_UP
```

**Description:**

This message is sent to indicate that the pointer is now up. *ui32Param1* is the X coordinate of the location where the pointer up event occurred, and *ui32Param2* is the Y coordinate. This message is delivered in bottom-up order.



### 4.2.2.10 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 `psParent` argument to a widget declaration) or as the `psWidget` argument to an API (such as [WidgetPaint\(\)](#) to paint the entire widget tree).

### 4.2.2.11 WidgetPaint

Requests a redraw of the widget tree.

**Definition:**

```
#define WidgetPaint(psWidget)
```

**Parameters:**

***psWidget*** 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 could 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 *psParent,  
          tWidget *psWidget)
```

**Parameters:**

***psParent*** is the parent for the widget. To add to the root of the tree set this parameter to **WIDGET\_ROOT**.

***psWidget*** 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 `psParent` field of each widget in the added tree is correctly set (in other words, only the widget pointed to by `psWidget` is updated to properly reside in the tree).

It is the responsibility of the caller to initialize the `psNext` and `psChild` 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:**

```
int32_t
WidgetDefaultMsgProc (tWidget *psWidget,
                     uint32_t ui32Message,
                     uint32_t ui32Param1,
                     uint32_t ui32Param2)
```

**Parameters:**

***psWidget*** is a pointer to the widget.

***ui32Message*** is the message to be processed.

***ui32Param1*** is the first parameter to the message.

***ui32Param2*** 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:**

```
int32_t
WidgetMessageQueueAdd (tWidget *psWidget,
                      uint32_t ui32Message,
                      uint32_t ui32Param1,
                      uint32_t ui32Param2,
                      bool bPostOrder,
                      bool bStopOnSuccess)
```

**Parameters:**

**psWidget** is the widget to which the message should be sent.

**ui32Message** is the message to be sent.

**ui32Param1** is the first parameter to the message.

**ui32Param2** 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:**

```
uint32_t
WidgetMessageSendPostOrder(tWidget *psWidget,
                           uint32_t ui32Message,
                           uint32_t ui32Param1,
```

```
uint32_t ui32Param2,  
bool bStopOnSuccess)
```

**Parameters:**

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

**ui32Message** is the message to send.

**ui32Param1** is the first parameter to the message.

**ui32Param2** is the second parameter to the message.

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

**Description:**

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

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

Special handling is performed for pointer-related messages. The widget that accepts **WIDGET\_MSG\_PTR\_DOWN** is remembered and subsequent **WIDGET\_MSG\_PTR\_MOVE** and **WIDGET\_MSG\_PTR\_UP** messages are sent directly to that widget.

**Returns:**

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

#### 4.2.3.6 WidgetMessageSendPreOrder

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

**Prototype:**

```
uint32_t  
WidgetMessageSendPreOrder(tWidget *psWidget,  
                           uint32_t ui32Message,  
                           uint32_t ui32Param1,  
                           uint32_t ui32Param2,  
                           bool bStopOnSuccess)
```

**Parameters:**

**psWidget** is a pointer to the widget tree.

**ui32Message** is the message to send.

**ui32Param1** is the first parameter to the message.

**ui32Param2** 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:**

```
uint32_t
WidgetMutexGet (uint8_t *pi8Mutex)
```

**Parameters:**

***pi8Mutex*** 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 (uint8_t *pi8Mutex)
```

**Parameters:**

***pi8Mutex*** 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 `pi8Mutex` 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 (uint8_t *pi8Mutex)
```

**Parameters:**

***pi8Mutex*** 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:**

```
int32_t  
WidgetPointerMessage (uint32_t ui32Message,  
                     int32_t i32X,  
                     int32_t i32Y)
```

**Parameters:**

***ui32Message*** is the pointer message to be sent.

***i32X*** is the X coordinate associated with the message.

***i32Y*** 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 *psWidget)
```

**Parameters:**

*psWidget* 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

Introduction .....	97
Definitions .....	97

### 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, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, 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`(psWidget)
- `CanvasAppDrawnOn`(psWidget)
- `CanvasCallbackSet`(psWidget, pfnOnPnt)
- `CanvasFillColorSet`(psWidget, ui32Color)
- `CanvasFillOff`(psWidget)
- `CanvasFillOn`(psWidget)
- `CanvasFontSet`(psWidget, pFnt)
- `CanvasImageOff`(psWidget)
- `CanvasImageOn`(psWidget)
- `CanvasImageSet`(psWidget, plmg)
- `CanvasOutlineColorSet`(psWidget, ui32Color)
- `CanvasOutlineOff`(psWidget)
- `CanvasOutlineOn`(psWidget)
- `CanvasStruct`(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pfnOnPaint)
- `CanvasTextAlignment`(psWidget, ui32Align)
- `CanvasTextColorSet`(psWidget, ui32Color)
- `CanvasTextOff`(psWidget)
- `CanvasTextOn`(psWidget)
- `CanvasTextOpaqueOff`(psWidget)
- `CanvasTextOpaqueOn`(psWidget)
- `CanvasTextSet`(psWidget, pcTxt)

## Functions

- void `CanvasInit` (tCanvasWidget \*psWidget, const tDisplay \*psDisplay, int32\_t i32X, int32\_t i32Y, int32\_t i32Width, int32\_t i32Height)
- int32\_t `CanvasMsgProc` (tWidget \*psWidget, uint32\_t ui32Msg, uint32\_t ui32Param1, uint32\_t ui32Param2)

## 5.2.1 Detailed Description

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

## 5.2.2 Data Structure Documentation

### 5.2.2.1 tCanvasWidget

#### Definition:

```
typedef struct
{
    tWidget sBase;
    uint32_t ui32Style;
    uint32_t ui32FillColor;
    uint32_t ui32OutlineColor;
    uint32_t ui32TextColor;
    const tFont *psFont;
    const char *pcText;
    const uint8_t *pui8Image;
    void (*pfnOnPaint)(tWidget *psWidget,
                      tContext *psContext);
}
tCanvasWidget
```

#### Members:

**sBase** The generic widget information.

**ui32Style** The style for this widget. This is a set of flags defined by `CANVAS_STYLE_xxx`.

**ui32FillColor** 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.

**ui32OutlineColor** The 24-bit RGB color used to outline this canvas, if `CANVAS_STYLE_OUTLINE` is selected.

**ui32TextColor** The 24-bit RGB color used to draw text on this canvas, if `CANVAS_STYLE_TEXT` is selected.

**psFont** A pointer to the font used to render the canvas text, if `CANVAS_STYLE_TEXT` is selected.

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

**pui8Image** A pointer to the image to be drawn onto this canvas, if `CANVAS_STYLE_IMG` is selected.

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

#### Description:

The structure that describes a canvas widget.

## 5.2.3 Define Documentation

### 5.2.3.1 Canvas

Declares an initialized variable containing a canvas widget data structure.

**Definition:**

```
#define Canvas(sName,  
              psParent,  
              psNext,  
              psChild,  
              psDisplay,  
              i32X,  
              i32Y,  
              i32Width,  
              i32Height,  
              ui32Style,  
              ui32FillColor,  
              ui32OutlineColor,  
              ui32TextColor,  
              psFont,  
              pcText,  
              pui8Image,  
              pfnOnPaint)
```

**Parameters:**

***sName*** is the name of the variable to be declared.

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

***psDisplay*** is a pointer to the display on which to draw the canvas.

***i32X*** is the X coordinate of the upper left corner of the canvas.

***i32Y*** is the Y coordinate of the upper left corner of the canvas.

***i32Width*** is the width of the canvas.

***i32Height*** is the height of the canvas.

***ui32Style*** is the style to be applied to the canvas.

***ui32FillColor*** is the color used to fill in the canvas.

***ui32OutlineColor*** is the color used to outline the canvas.

***ui32TextColor*** is the color used to draw text on the canvas.

***psFont*** 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.

***pui8Image*** 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).

***ui32Style*** 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 *psFont* and *pcText*).
- **CANVAS\_STYLE\_IMG** to indicate that the canvas should have an image drawn on it (using *pui8Image*).
- **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(psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget,  
                          pfnOnPnt)
```

**Parameters:**

***psWidget*** 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(psWidget,  
                          ui32Color)
```



**Parameters:**

*psWidget* is a pointer to the canvas widget to be modified.

*ui32Color* 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(psWidget)
```

**Parameters:**

*psWidget* 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(psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget,  
                      pFnt)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                        pImg)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                               ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the canvas widget to be modified.

***ui32Color*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget)
```

**Parameters:**  
***psWidget*** 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(psParent,  
                    psNext,  
                    psChild,  
                    psDisplay,  
                    i32X,  
                    i32Y,  
                    i32Width,  
                    i32Height,  
                    ui32Style,  
                    ui32FillColor,  
                    ui32OutlineColor,  
                    ui32TextColor,  
                    psFont,  
                    pcText,  
                    pui8Image,  
                    pfnOnPaint)
```

**Parameters:**  
***psParent*** is a pointer to the parent widget.  
***psNext*** is a pointer to the sibling widget.  
***psChild*** is a pointer to the first child widget.  
***psDisplay*** is a pointer to the display on which to draw the canvas.  
***i32X*** is the X coordinate of the upper left corner of the canvas.  
***i32Y*** is the Y coordinate of the upper left corner of the canvas.  
***i32Width*** is the width of the canvas.

***i32Height*** is the height of the canvas.  
***ui32Style*** is the style to be applied to the canvas.  
***ui32FillColor*** is the color used to fill in the canvas.  
***ui32OutlineColor*** is the color used to outline the canvas.  
***ui32TextColor*** is the color used to draw text on the canvas.  
***psFont*** 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.  
***pui8Image*** 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(...)
};
```

***ui32Style*** 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 *psFont* and *pcText*).
- **CANVAS\_STYLE\_IMG** to indicate that the canvas should have an image drawn on it (using *pui8Image*).
- **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 (psWidget,  
                             ui32Align)
```

**Parameters:**

**psWidget** is a pointer to the canvas widget to modify.

**ui32Align** 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 bounding 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 (psWidget,  
                             ui32Color)
```

**Parameters:**

**psWidget** is a pointer to the canvas widget to be modified.

**ui32Color** 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 (psWidget)
```

**Parameters:**

**psWidget** 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(psWidget)
```

**Parameters:**

*psWidget* 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(psWidget)
```

**Parameters:**

*psWidget* 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(psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget,  
                      pcTxt)
```

**Parameters:**

*psWidget* 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 *psWidget,  
            const tDisplay *psDisplay,  
            int32_t i32X,  
            int32_t i32Y,  
            int32_t i32Width,  
            int32_t i32Height)
```

**Parameters:**

*psWidget* is a pointer to the canvas widget to initialize.

*psDisplay* is a pointer to the display on which to draw the canvas.

*i32X* is the X coordinate of the upper left corner of the canvas.

*i32Y* is the Y coordinate of the upper left corner of the canvas.



*i32Width* is the width of the canvas.

*i32Height* 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:**

```
int32_t  
CanvasMsgProc(tWidget *psWidget,  
              uint32_t ui32Msg,  
              uint32_t ui32Param1,  
              uint32_t ui32Param2)
```

**Parameters:**

*psWidget* is a pointer to the canvas widget.

*ui32Msg* is the message.

*ui32Param1* is the first parameter to the message.

*ui32Param2* 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

Introduction .....	115
Definitions .....	115

### 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, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui16Style, ui16BoxSize, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pfnOnChange)
- [CheckBoxBoxSizeSet](#)(psWidget, ui16Size)
- [CheckBoxCallbackSet](#)(psWidget, pfnOnChg)
- [CheckBoxFillColorSet](#)(psWidget, ui32Color)
- [CheckBoxFillOff](#)(psWidget)
- [CheckBoxFillOn](#)(psWidget)
- [CheckBoxFontSet](#)(psWidget, pFnt)
- [CheckBoxImageOff](#)(psWidget)
- [CheckBoxImageOn](#)(psWidget)
- [CheckBoxImageSet](#)(psWidget, plmg)
- [CheckBoxOutlineColorSet](#)(psWidget, ui32Color)
- [CheckBoxOutlineOff](#)(psWidget)
- [CheckBoxOutlineOn](#)(psWidget)
- [CheckBoxStruct](#)(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui16Style, ui16BoxSize, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pfnOnChange)
- [CheckBoxTextColorSet](#)(psWidget, ui32Color)
- [CheckBoxTextOff](#)(psWidget)
- [CheckBoxTextOn](#)(psWidget)
- [CheckBoxTextOpaqueOff](#)(psWidget)
- [CheckBoxTextOpaqueOn](#)(psWidget)
- [CheckBoxTextSet](#)(psWidget, pcTxt)

## Functions

- void [CheckBoxInit](#) (tCheckBoxWidget \*psWidget, const tDisplay \*psDisplay, int32\_t i32X, int32\_t i32Y, int32\_t i32Width, int32\_t i32Height)
- int32\_t [CheckBoxMsgProc](#) (tWidget \*psWidget, uint32\_t ui32Msg, uint32\_t ui32Param1, uint32\_t ui32Param2)

### 6.2.1 Detailed Description

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

## 6.2.2 Data Structure Documentation

### 6.2.2.1 tCheckBoxWidget

#### Definition:

```
typedef struct
{
    tWidget sBase;
    uint16_t ui16Style;
    uint16_t ui16BoxSize;
    uint32_t ui32FillColor;
    uint32_t ui32OutlineColor;
    uint32_t ui32TextColor;
    const tFont *psFont;
    const char *pcText;
    const uint8_t *pui8Image;
    void (*pfnOnChange)(tWidget *psWidget,
                       uint32_t bSelected);
}
tCheckBoxWidget
```

#### Members:

**sBase** The generic widget information.

**ui16Style** The style for this check box. This is a set of flags defined by CB\_STYLE\_XXX.

**ui16BoxSize** 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).

**ui32FillColor** 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.

**ui32OutlineColor** The 24-bit RGB color used to outline this check box, if CB\_STYLE\_OUTLINE is selected.

**ui32TextColor** The 24-bit RGB color used to draw text on this check box, if CB\_STYLE\_TEXT is selected.

**psFont** 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.

**pui8Image** 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,
                psParent,
                psNext,
                psChild,
                psDisplay,
                i32X,
                i32Y,
                i32Width,
                i32Height,
                ui16Style,
                ui16BoxSize,
                ui32FillColor,
                ui32OutlineColor,
                ui32TextColor,
                psFont,
                pcText,
                pui8Image,
                pfnOnChange)
```

**Parameters:**

- sName*** is the name of the variable to be declared.
- psParent*** is a pointer to the parent widget.
- psNext*** is a pointer to the sibling widget.
- psChild*** is a pointer to the first child widget.
- psDisplay*** is a pointer to the display on which to draw the check box.
- i32X*** is the X coordinate of the upper left corner of the check box.
- i32Y*** is the Y coordinate of the upper left corner of the check box.
- i32Width*** is the width of the check box.
- i32Height*** is the height of the check box.
- ui16Style*** is the style to be applied to this check box.
- ui16BoxSize*** is the size of the box that is checked.
- ui32FillColor*** is the color used to fill in the check box.
- ui32OutlineColor*** is the color used to outline the check box.
- ui32TextColor*** is the color used to draw text on the check box.
- psFont*** 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.
- pui8Image*** 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).

*ui16Style* 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 *psFont* and *pcText*).
- **CB\_STYLE\_IMG** to indicate that the check box should have an image drawn on it (using *pui8Image*).
- **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 (psWidget,  
                           ui16Size)
```

**Parameters:**

***psWidget*** is a pointer to the check box widget to modify.

***ui16Size*** 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 (psWidget,  
                             pfnOnChg)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                             ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the check box widget to be modified.  
*ui32Color* 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 (psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget,  
                        pFnt)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                          pImg)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                                ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the check box widget to be modified.

***ui32Color*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget)`

**Parameters:**  
***psWidget*** 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(psParent,  
                          psNext,  
                          psChild,  
                          psDisplay,  
                          i32X,  
                          i32Y,  
                          i32Width,  
                          i32Height,  
                          ui16Style,  
                          ui16BoxSize,  
                          ui32FillColor,  
                          ui32OutlineColor,  
                          ui32TextColor,  
                          psFont,  
                          pcText,  
                          pui8Image,  
                          pfnOnChange)`

**Parameters:**  
***psParent*** is a pointer to the parent widget.  
***psNext*** is a pointer to the sibling widget.  
***psChild*** is a pointer to the first child widget.  
***psDisplay*** is a pointer to the display on which to draw the check box.  
***i32X*** is the X coordinate of the upper left corner of the check box.  
***i32Y*** is the Y coordinate of the upper left corner of the check box.

***i32Width*** is the width of the check box.  
***i32Height*** is the height of the check box.  
***ui16Style*** is the style to be applied to this check box.  
***ui16BoxSize*** is the size of the box that is checked.  
***ui32FillColor*** is the color used to fill in the check box.  
***ui32OutlineColor*** is the color used to outline the check box.  
***ui32TextColor*** is the color used to draw text on the check box.  
***psFont*** 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.  
***pui8Image*** 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(...)
};
```

***ui16Style*** 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 *psFont* and *pcText*).
- **CB\_STYLE\_IMG** to indicate that the check box should have an image drawn on it (using *pui8Image*).
- **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(psWidget,
                             ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the check box widget to be modified.

***ui32Color*** 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(psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget,  
                        pcTxt)
```

**Parameters:**

*psWidget* 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 *psWidget,  
             const tDisplay *psDisplay,  
             int32_t i32X,  
             int32_t i32Y,  
             int32_t i32Width,  
             int32_t i32Height)
```

**Parameters:**

***psWidget*** is a pointer to the check box widget to initialize.

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

***i32X*** is the X coordinate of the upper left corner of the check box.

***i32Y*** is the Y coordinate of the upper left corner of the check box.

***i32Width*** is the width of the check box.

***i32Height*** 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:**

```
int32_t  
CheckBoxMsgProc (tWidget *psWidget,  
               uint32_t ui32Msg,  
               uint32_t ui32Param1,  
               uint32_t ui32Param2)
```

**Parameters:**

***psWidget*** is a pointer to the check box widget.

***ui32Msg*** is the message.

***ui32Param1*** is the first parameter to the message.

***ui32Param2*** 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

Introduction .....	131
Definitions .....	131

## 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, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText)
- **ContainerFillColorSet**(psWidget, ui32Color)
- **ContainerFillOff**(psWidget)
- **ContainerFillOn**(psWidget)
- **ContainerFontSet**(psWidget, pFnt)
- **ContainerOutlineColorSet**(psWidget, ui32Color)

- [ContainerOutlineOff](#)(psWidget)
- [ContainerOutlineOn](#)(psWidget)
- [ContainerStruct](#)(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText)
- [ContainerTextCenterOff](#)(psWidget)
- [ContainerTextCenterOn](#)(psWidget)
- [ContainerTextColorSet](#)(psWidget, ui32Color)
- [ContainerTextOff](#)(psWidget)
- [ContainerTextOn](#)(psWidget)
- [ContainerTextOpaqueOff](#)(psWidget)
- [ContainerTextOpaqueOn](#)(psWidget)
- [ContainerTextSet](#)(psWidget, pcTxt)
- [CTR\\_STYLE\\_FILL](#)
- [CTR\\_STYLE\\_OUTLINE](#)
- [CTR\\_STYLE\\_TEXT](#)
- [CTR\\_STYLE\\_TEXT\\_CENTER](#)
- [CTR\\_STYLE\\_TEXT\\_OPAQUE](#)

## Functions

- void [ContainerInit](#) ([tContainerWidget](#) \*psWidget, const [tDisplay](#) \*psDisplay, [int32\\_t](#) i32X, [int32\\_t](#) i32Y, [int32\\_t](#) i32Width, [int32\\_t](#) i32Height)
- [int32\\_t](#) [ContainerMsgProc](#) ([tWidget](#) \*psWidget, [uint32\\_t](#) ui32Msg, [uint32\\_t](#) ui32Param1, [uint32\\_t](#) ui32Param2)

### 7.2.1 Detailed Description

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

### 7.2.2 Data Structure Documentation

#### 7.2.2.1 tContainerWidget

**Definition:**

```
typedef struct
{
    tWidget sBase;
    uint32\_t ui32Style;
    uint32\_t ui32FillColor;
    uint32\_t ui32OutlineColor;
    uint32\_t ui32TextColor;
    const tFont *psFont;
    const char *pcText;
}
tContainerWidget
```

**Members:**

**sBase** The generic widget information.

**ui32Style** The style for this widget. This is a set of flags defined by CTR\_STYLE\_XXX.

**ui32FillColor** 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.

**ui32OutlineColor** The 24-bit RGB color used to outline this container widget, if CTR\_STYLE\_OUTLINE is selected.

**ui32TextColor** The 24-bit RGB color used to draw text on this container widget, if CTR\_STYLE\_TEXT is selected.

**psFont** 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,
                psParent,
                psNext,
                psChild,
                psDisplay,
                i32X,
                i32Y,
                i32Width,
                i32Height,
                ui32Style,
                ui32FillColor,
                ui32OutlineColor,
                ui32TextColor,
                psFont,
                pcText)
```

**Parameters:**

**sName** is the name of the variable to be declared.

**psParent** is a pointer to the parent widget.

**psNext** is a pointer to the sibling widget.

**psChild** is a pointer to the first child widget.

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

**i32X** is the X coordinate of the upper left corner of the container widget.

**i32Y** is the Y coordinate of the upper left corner of the container widget.

**i32Width** is the width of the container widget.

**i32Height** is the height of the container widget.

***ui32Style*** is the style to be applied to the container widget.

***ui32FillColor*** is the color used to fill in the container widget.

***ui32OutlineColor*** is the color used to outline the container widget.

***ui32TextColor*** is the color used to draw text on the container widget.

***psFont*** 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).

*ui32Style* 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 *psFont* 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 (psWidget,  
                             ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the container widget to be modified.

***ui32Color*** 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(psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget,  
                        pFnt)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                                ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the container widget to be modified.  
***ui32Color*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psParent,
                        psNext,
                        psChild,
                        psDisplay,
                        i32X,
                        i32Y,
                        i32Width,
                        i32Height,
                        ui32Style,
                        ui32FillColor,
                        ui32OutlineColor,
                        ui32TextColor,
                        psFont,
                        pcText)
```

**Parameters:**

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

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

***i32X*** is the X coordinate of the upper left corner of the container widget.

***i32Y*** is the Y coordinate of the upper left corner of the container widget.

***i32Width*** is the width of the container widget.

***i32Height*** is the height of the container widget.

***ui32Style*** is the style to be applied to the container widget.

***ui32FillColor*** is the color used to fill in the container widget.

***ui32OutlineColor*** is the color used to outline the container widget.

***ui32TextColor*** is the color used to draw text on the container widget.

***psFont*** 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 (...),
};
```

***ui32Style*** 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 *psFont* 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(psWidget)
```

**Parameters:**

***psWidget*** 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(psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                             ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the container widget to be modified.

***ui32Color*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                        pcTxt)
```

**Parameters:**

***psWidget*** 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 *psWidget,  
              const tDisplay *psDisplay,  
              int32_t i32X,  
              int32_t i32Y,  
              int32_t i32Width,  
              int32_t i32Height)
```

**Parameters:**

***psWidget*** is a pointer to the container widget to initialize.

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

***i32X*** is the X coordinate of the upper left corner of the container widget.

***i32Y*** is the Y coordinate of the upper left corner of the container widget.

***i32Width*** is the width of the container widget.

***i32Height*** 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:**

```
int32_t  
ContainerMsgProc (tWidget *psWidget,  
                 uint32_t ui32Msg,  
                 uint32_t ui32Param1,  
                 uint32_t ui32Param2)
```

**Parameters:**

***psWidget*** is a pointer to the container widget.

***ui32Msg*** is the message.

***ui32Param1*** is the first parameter to the message.

***ui32Param2*** 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

Introduction .....	145
Definitions .....	145

### 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, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32ForeColor, ui32PressColor, ui32BackColor, psFont, pcText, pui8Image, pui8PressImage, pui8KeycapImage, i16XOff, i16YOff, ui16AutoRepeatDelay, ui16AutoRepeatRate, pfnOnClick)
- `ImageButtonAutoRepeatDelaySet`(psWidget, ui16Delay)
- `ImageButtonAutoRepeatOff`(psWidget)
- `ImageButtonAutoRepeatOn`(psWidget)
- `ImageButtonAutoRepeatRateSet`(psWidget, ui16Rate)
- `ImageButtonBackgroundColorSet`(psWidget, ui32Color)
- `ImageButtonCallbackSet`(psWidget, pfnOnClik)
- `ImageButtonFillColorSet`(psWidget, ui32Color)
- `ImageButtonFillOff`(psWidget)
- `ImageButtonFillOn`(psWidget)
- `ImageButtonForegroundColorSet`(psWidget, ui32Color)
- `ImageButtonImageKeycapSet`(psWidget, plmg)
- `ImageButtonImageOff`(psWidget)
- `ImageButtonImageOn`(psWidget)
- `ImageButtonImagePressedSet`(psWidget, plmg)
- `ImageButtonImageSet`(psWidget, plmg)
- `ImageButtonKeycapOff`(psWidget)
- `ImageButtonKeycapOffsetSet`(psWidget, i16X, i16Y)
- `ImageButtonKeycapOn`(psWidget)
- `ImageButtonPressedColorSet`(psWidget, ui32Color)
- `ImageButtonStruct`(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32ForeColor, ui32PressColor, ui32BackColor, psFont, pcText, pui8Image, pui8PressImage, pui8KeycapImage, i16XOff, i16YOff, ui16AutoRepeatDelay, ui16AutoRepeatRate, pfnOnClick)
- `ImageButtonTextOff`(psWidget)
- `ImageButtonTextOn`(psWidget)
- `ImageButtonTextSet`(psWidget, pcTxt)

## Functions

- void `ImageButtonInit` (`tImageButtonWidget` \*psWidget, const `tDisplay` \*psDisplay, int32\_t i32X, int32\_t i32Y, int32\_t i32Width, int32\_t i32Height)
- int32\_t `ImageButtonMsgProc` (`tWidget` \*psWidget, uint32\_t ui32Msg, uint32\_t ui32Param1, uint32\_t ui32Param2)

### 8.2.1 Detailed Description

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

## 8.2.2 Data Structure Documentation

### 8.2.2.1 tImageButtonWidget

#### Definition:

```
typedef struct
{
    tWidget sBase;
    uint32_t ui32Style;
    uint32_t ui32ForegroundColor;
    uint32_t ui32PressedColor;
    uint32_t ui32BackgroundColor;
    const tFont *psFont;
    const char *pcText;
    const uint8_t *pui8Image;
    const uint8_t *pui8PressImage;
    const uint8_t *pui8KeycapImage;
    int16_t i16XOffset;
    int16_t i16YOffset;
    uint16_t ui16AutoRepeatDelay;
    uint16_t ui16AutoRepeatRate;
    uint32_t ui32AutoRepeatCount;
    void (*pfnOnClick)(tWidget *psWidget);
}
tImageButtonWidget
```

#### Members:

**sBase** The generic widget information.

**ui32Style** The style for this widget. This is a set of flags defined by IB\_STYLE\_XXX.

**ui32ForegroundColor** 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.

**ui32PressedColor** 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.

**ui32BackgroundColor** 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 unpressed state.

**psFont** 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.

**pui8Image** A pointer to the image to be drawn onto this image button, if IB\_STYLE\_IMG is selected.

**pui8PressImage** A pointer to the image to be drawn onto this image button when it is pressed.

**pui8KeycapImage** A pointer to the image to be drawn above the background image for the button.

**i16XOffset** The number of pixels to move the keycap image horizontally when the button is drawn in its pressed state.

**i16YOffset** The number of pixels to move the keycap image vertically when the button is drawn in its pressed state.

**ui16AutoRepeatDelay** 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.

**ui16AutoRepeatRate** 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.

**ui32AutoRepeatCount** 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,  
                    psParent,  
                    psNext,  
                    psChild,  
                    psDisplay,  
                    i32X,  
                    i32Y,  
                    i32Width,  
                    i32Height,  
                    ui32Style,  
                    ui32ForeColor,  
                    ui32PressColor,  
                    ui32BackColor,  
                    psFont,
```

```
pcText,  
pui8Image,  
pui8PressImage,  
pui8KeycapImage,  
i16XOff,  
i16YOff,  
ui16AutoRepeatDelay,  
ui16AutoRepeatRate,  
pfnOnClick)
```

**Parameters:**

**sName** is the name of the variable to be declared.

**psParent** is a pointer to the parent widget.

**psNext** is a pointer to the sibling widget.

**psChild** is a pointer to the first child widget.

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

**i32X** is the X coordinate of the upper left corner of the image button.

**i32Y** is the Y coordinate of the upper left corner of the image button.

**i32Width** is the width of the image button.

**i32Height** is the height of the image button.

**ui32Style** is the style to be applied to the image button.

**ui32ForeColor** 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.

**ui32PressColor** 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.

**ui32BackColor** 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.

**psFont** 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.

**pui8Image** is a pointer to the image to draw on the background of this image button when it is in the released state.

**pui8PressImage** is a pointer to the image to draw on the background of this image button when it is in the pressed state.

**pui8KeycapImage** 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.

**i16XOff** is the horizontal offset to apply when drawing the keycap image on the button when in the pressed state.

**i16YOff** is the vertical offset to apply when drawing the keycap image on the button when in the pressed state.

**ui16AutoRepeatDelay** is the delay before starting auto-repeat.

**ui16AutoRepeatRate** 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).

*ui32Style* 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 (psWidget,  
                                       ui16Delay)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to modify.

***ui16Delay*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                                     ui16Rate)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to modify.

***ui16Rate*** 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 (psWidget,  
                                     ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to be modified.

***ui32Color*** 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 (psWidget,  
                               pfnOnClick)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to modify.  
***pfnOnClick*** 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 (psWidget,  
                               ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to be modified.  
***ui32Color*** 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 (psWidget)
```

**Parameters:**

*psWidget* 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(psWidget)
```

**Parameters:**

*psWidget* 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(psWidget,  
                                     ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the image button widget to be modified.

*ui32Color* 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 (psWidget,  
                                   pImg)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                                   pImg)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                             pImg)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                                   i16X,  
                                   i16Y)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to be modified.

***i16X*** is the signed horizontal position offset for the keycap image when the image button is pressed. Positive values move the image right.

***i16Y*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                                   ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to be modified.  
***ui32Color*** 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 (psParent,  
                          psNext,  
                          psChild,  
                          psDisplay,  
                          i32X,  
                          i32Y,  
                          i32Width,  
                          i32Height,  
                          ui32Style,  
                          ui32ForeColor,  
                          ui32PressColor,  
                          ui32BackColor,  
                          psFont,  
                          pcText,  
                          pui8Image,  
                          pui8PressImage,  
                          pui8KeycapImage,  
                          i16XOff,  
                          i16YOff,  
                          ui16AutoRepeatDelay,  
                          ui16AutoRepeatRate,  
                          pfnOnClick)
```

**Parameters:**

***psParent*** is a pointer to the parent widget.  
***psNext*** is a pointer to the sibling widget.  
***psChild*** is a pointer to the first child widget.  
***psDisplay*** is a pointer to the display on which to draw the push button.  
***i32X*** is the X coordinate of the upper left corner of the image button.  
***i32Y*** is the Y coordinate of the upper left corner of the image button.

***i32Width*** is the width of the image button.

***i32Height*** is the height of the image button.

***ui32Style*** is the style to be applied to the image button.

***ui32ForeColor*** 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.

***ui32PressColor*** 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.

***ui32BackColor*** 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.

***psFont*** 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.

***pui8Image*** is a pointer to the image to draw on the background of this image button when it is in the released state.

***pui8PressImage*** is a pointer to the image to draw on the background of this image button when it is in the pressed state.

***pui8KeycapImage*** 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.

***i16XOff*** is the horizontal offset to apply when drawing the keycap image on the button when in the pressed state.

***i16YOff*** is the vertical offset to apply when drawing the keycap image on the button when in the pressed state.

***ui16AutoRepeatDelay*** is the delay before starting auto-repeat.

***ui16AutoRepeatRate*** 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(...)
};
```

***ui32Style*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                             pcTxt)
```

**Parameters:**

***psWidget*** 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 (tImageWidget *psWidget,  
                 const tDisplay *psDisplay,  
                 int32_t i32X,  
                 int32_t i32Y,  
                 int32_t i32Width,  
                 int32_t i32Height)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget to initialize.

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

***i32X*** is the X coordinate of the upper left corner of the image button.

***i32Y*** is the Y coordinate of the upper left corner of the image button.

***i32Width*** is the width of the image button.

***i32Height*** 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:**

```
int32_t  
ImageButtonMsgProc (tWidget *psWidget,  
                   uint32_t ui32Msg,  
                   uint32_t ui32Param1,  
                   uint32_t ui32Param2)
```

**Parameters:**

***psWidget*** is a pointer to the image button widget.

***ui32Msg*** is the message.

***ui32Param1*** is the first parameter to the message.

***ui32Param2*** 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

Introduction .....	163
Definitions .....	164

### 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 outputting 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, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32BgColor, ui32SelBgColor, ui32TextColor, ui32SelTextColor, ui32OutlineColor, psFont, ppcText, ui16MaxEntries, ui16PopulatedEntries, pfnOnChange)`
- `LISTBOX_STYLE_LOCKED`
- `LISTBOX_STYLE_OUTLINE`
- `LISTBOX_STYLE_WRAP`
- `ListboxBackgroundColorSet(psWidget, ui32Color)`
- `ListboxCallbackSet(psWidget, pfnCallback)`
- `ListboxClear(psWidget)`
- `ListboxFontSet(psWidget, pFnt)`
- `ListboxLock(psWidget)`
- `ListboxOutlineColorSet(psWidget, ui32Color)`
- `ListboxOutlineOff(psWidget)`
- `ListboxOutlineOn(psWidget)`
- `ListboxSelectedBackgroundColorSet(psWidget, ui32Color)`
- `ListboxSelectedTextColorSet(psWidget, ui32Color)`
- `ListboxSelectionGet(psWidget)`
- `ListboxSelectionSet(psWidget, i16Sel)`
- `ListboxStruct(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32BgColor, ui32SelBgColor, ui32TextColor, ui32SelTextColor, ui32OutlineColor, psFont, ppcText, ui16MaxEntries, ui16PopulatedEntries, pfnOnChange)`
- `ListboxTextColorSet(psWidget, ui32Color)`
- `ListboxTextSet(psWidget, pcTxt, ui32Index)`
- `ListboxUnlock(psWidget)`
- `ListboxWrapDisable(psWidget)`
- `ListboxWrapEnable(psWidget)`

## Functions

- void `ListBoxInit` (`tListBoxWidget *psWidget`, const `tDisplay *psDisplay`, const char `**ppcText`, `uint16_t ui16MaxEntries`, `uint16_t ui16PopulatedEntries`, `int32_t i32X`, `int32_t i32Y`, `int32_t i32Width`, `int32_t i32Height`)
- `int32_t` `ListBoxMsgProc` (`tWidget *psWidget`, `uint32_t ui32Msg`, `uint32_t ui32Param1`, `uint32_t ui32Param2`)
- `int32_t` `ListBoxTextAdd` (`tListBoxWidget *pListBox`, const char `*pcTxt`)

### 9.2.1 Detailed Description

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

### 9.2.2 Data Structure Documentation

#### 9.2.2.1 tListBoxWidget

##### Definition:

```
typedef struct
{
    tWidget sBase;
    uint32_t ui32Style;
    uint32_t ui32BackgroundColor;
    uint32_t ui32SelectedBackgroundColor;
    uint32_t ui32TextColor;
    uint32_t ui32SelectedTextColor;
    uint32_t ui32OutlineColor;
    const tFont *psFont;
    const char **ppcText;
    uint16_t ui16MaxEntries;
    uint16_t ui16Populated;
    int16_t i16Selected;
    uint16_t ui16StartEntry;
    uint16_t ui16OldestEntry;
    uint16_t ui16Scrolled;
    int32_t i32PointerY;
    void (*pfnOnChange)(tWidget *psWidget,
                       int16_t ui16SelIndex);
}
tListBoxWidget
```

##### Members:

***sBase*** The generic widget information.

***ui32Style*** The style for this widget. This is a set of flags defined by `LISTBOX_STYLE_XXX`.

***ui32BackgroundColor*** The 24-bit RGB color used as the background for the listbox.

***ui32SelectedBackgroundColor*** The 24-bit RGB color used as the background for the selected entry in the listbox.

***ui32TextColor*** The 24-bit RGB color used to draw text on this listbox.

- ui32SelectedTextColor** The 24-bit RGB color used to draw the selected text on this listbox.
- ui32OutlineColor** The 24-bit RGB color used to outline this listbox, if LISTBOX\_STYLE\_OUTLINE is selected.
- psFont** 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.
- ui16MaxEntries** The number of elements in the array pointed to by ppcText.
- ui16Populated** The number of elements in the array pointed to by ppcText which are currently populated with strings.
- i16Selected** The index of the string currently selected in the list box. If no selection has been made, this will be set to 0xFFFF (-1).
- ui16StartEntry** 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.
- ui16OldestEntry** 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.
- ui16Scrolled** 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.
- i32PointerY** 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 ui16SelIndex 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,
                psParent,
                psNext,
                psChild,
                psDisplay,
                i32X,
                i32Y,
                i32Width,
                i32Height,
                ui32Style,
```

```

    ui32BgColor,
    ui32SelBgColor,
    ui32TextColor,
    ui32SelTextColor,
    ui32OutlineColor,
    psFont,
    ppcText,
    ui16MaxEntries,
    ui16PopulatedEntries,
    pfnOnChange)

```

**Parameters:**

***sName*** is the name of the variable to be declared.

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

***psDisplay*** is a pointer to the display on which to draw the listbox.

***i32X*** is the X coordinate of the upper left corner of the listbox.

***i32Y*** is the Y coordinate of the upper left corner of the listbox.

***i32Width*** is the width of the listbox.

***i32Height*** is the height of the listbox.

***ui32Style*** is the style to be applied to the listbox.

***ui32BgColor*** is the background color for the listbox.

***ui32SelBgColor*** is the background color for the selected element in the listbox.

***ui32TextColor*** is the color used to draw text on the listbox.

***ui32SelTextColor*** is the color used to draw the selected element text in the listbox.

***ui32OutlineColor*** is the color used to outline the listbox.

***psFont*** 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.

***ui16MaxEntries*** provides the number of entries in the *ppcText* array and represents the maximum number of strings the listbox can hold.

***ui16PopulatedEntries*** 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).

*ui32Style* 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 `ui32OutlineColor` field of the widget structure and the inner rectangle in color `ui32BackgroundColor`.

### 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 (psWidget,  
                                ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the listbox widget to be modified.  
***ui32Color*** 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 (psWidget,  
                           pfnCallback)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                      pFnt)
```

**Parameters:**

***psWidget*** 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(psWidget)
```

**Parameters:**

*psWidget* 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(psWidget,  
                               ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the listbox widget to be modified.

*ui32Color* 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 (psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget)
```

**Parameters:**

*psWidget* 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 (psWidget,  
                                         ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the listbox widget to be modified.

*ui32Color* 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 (psWidget,  
                                   ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the listbox widget to be modified.

***ui32Color*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget,  
                             i16Sel)
```

**Parameters:**

***psWidget*** is a pointer to the listbox widget to modify.

***i16Sel*** 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 (psParent,
                    psNext,
                    psChild,
                    psDisplay,
                    i32X,
                    i32Y,
                    i32Width,
                    i32Height,
                    ui32Style,
                    ui32BgColor,
                    ui32SelBgColor,
                    ui32TextColor,
                    ui32SelTextColor,
                    ui32OutlineColor,
                    psFont,
                    ppcText,
                    ui16MaxEntries,
                    ui16PopulatedEntries,
                    pfnOnChange)
```

**Parameters:**

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

***psDisplay*** is a pointer to the display on which to draw the listbox.

***i32X*** is the X coordinate of the upper left corner of the listbox.

***i32Y*** is the Y coordinate of the upper left corner of the listbox.

***i32Width*** is the width of the listbox.

***i32Height*** is the height of the listbox.

***ui32Style*** is the style to be applied to the listbox.

***ui32BgColor*** is the background color for the listbox.

***ui32SelBgColor*** is the background color for the selected element in the listbox.

***ui32TextColor*** is the color used to draw text on the listbox.

***ui32SelTextColor*** is the color used to draw the selected element text in the listbox.

***ui32OutlineColor*** is the color used to outline the listbox.

***psFont*** 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.

***ui16MaxEntries*** provides the number of entries in the *ppcText* array and represents the maximum number of strings the listbox can hold.

***ui16PopulatedEntries*** indicates the number of entries in the *ppcText* array that currently hold valid string for the listbox.

***pfonChange*** 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(...)
};
```

*ui32Style* 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 (psWidget,
                             ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the listbox widget to be modified.  
***ui32Color*** 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 (psWidget,  
                      pcTxt,  
                      ui32Index)
```

**Parameters:**

***psWidget*** is a pointer to the listbox widget to be modified.

***pcTxt*** is a pointer to the new text string.

***ui32Index*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** 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 (psWidget)
```

**Parameters:**

***psWidget*** is a pointer to the listbox widget to modify.

**Description:**

This function enables 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.

## 9.2.4 Function Documentation

### 9.2.4.1 ListBoxInit

Initializes a listbox widget.

**Prototype:**

```
void
ListBoxInit (tListBoxWidget *psWidget,
             const tDisplay *psDisplay,
             const char **ppcText,
             uint16_t ui16MaxEntries,
             uint16_t ui16PopulatedEntries,
             int32_t i32X,
             int32_t i32Y,
             int32_t i32Width,
             int32_t i32Height)
```

**Parameters:**

***psWidget*** is a pointer to the listbox widget to initialize.

***psDisplay*** 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.

***ui16MaxEntries*** provides the total number of entries in the *ppcText* array.

***ui16PopulatedEntries*** provides the number of entries in the *ppcText* array which are populated.

***i32X*** is the X coordinate of the upper left corner of the listbox.

***i32Y*** is the Y coordinate of the upper left corner of the listbox.

***i32Width*** is the width of the listbox.

***i32Height*** 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:**

```
int32_t  
ListBoxMsgProc(tWidget *psWidget,  
               uint32_t ui32Msg,  
               uint32_t ui32Param1,  
               uint32_t ui32Param2)
```

**Parameters:**

**psWidget** is a pointer to the listbox widget.

**ui32Msg** is the message.

**ui32Param1** is the first parameter to the message.

**ui32Param2** 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:**

```
int32_t  
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 Keyboard Widget

Introduction .....	179
Definitions .....	180

## 10.1 Introduction

The keyboard widget allows the user create an on-screen keyboard for entering text without an external keyboard. The touch screen can be used to handle the pointer for selecting keys. Whenever a key is pressed a message is sent to the application callback to allow the application to handle the newly pressed key. The keyboard widget does not handle printing any of the keys as they are pressed, leaving all processing of keys to the application.

The keyboard can be defined dynamically using the `KeyboardInit()` function or defined using the `Keyboard()` macro provided by the graphics library.

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

- The keyboard background is filled with its fill color if the keyboard background style is selected by specifying the `KEYBOARD_STYLE_BG` flag.
- The keys are then draw according to their size specified in the `tKeyImage` or `tKeyText` structure. The sizes and position are specified in percentages in multiples of 100. This means that 100 represents 1% of the keyboard area and 10000 represents 100% of the screen.
- If the keyboard has outlining enabled, by specifying the `KEYBOARD_STYLE_OUTLINE` flag, then each key is outlined with its outline color.
- If the keyboard has fill enabled, by specifying the `KEYBOARD_STYLE_FILL` flag, then each key is filled with its fill color.
- If the keyboard is an image keyboard, by specifying the `KEYBOARD_STYLE_IMG` flag, then the image provided for each key is drawn in the middle each key. An image must be provided for all keys.
- If the keyboard is an text keyboard, by specifying the `KEYBOARD_STYLE_TEXT` flag, then the unicode character is drawn in the middle each key.

If the application needs auto-repeat of pressed keys it can provide the `KEYBOARD_STYLE_AUTO_REPEAT` optional flag and provide auto-repeat delay and repeat rate times for the keyboard. The keyboard can also provide release notification if the `KEYBOARD_STYLE_RELEASE_NOTIFY` is specified.

The graphics library provides a pre-defined keyboard for a US English keyboard but an application can provide its own keyboard in any form that the application requires. The default keyboard is defined in the `g_psKeyboardUSEnglish` variable.

The following example shows a sample keyboard definition for the default US English keyboard provided by the graphics library. The keyboard is located at the x,y location of 8,90 and is 300 pixels wide and 140 pixels high. The keyboard has a background fill color of `ClrBlack` enabled. The keys have a `ClrDarkGray` background when released and a `ClrGray` background when pressed with

ClrWhite text. The keyboard also has auto-repeat enabled with a delay of 100 events and a repeat rate of 50 events. The KeyPress() function is called for all key press events.

Example: Keyboard definition

```
Keyboard(g_sKeyboard, 0, 0, 0,
        &g_sKentec320x240x16_SSD2119, 8, 90, 300, 140,
        KEYBOARD_STYLE_TEXT | KEYBOARD_STYLE_FILL |
        KEYBOARD_STYLE_AUTO_REPEAT | KEYBOARD_STYLE_BG,
        ClrBlack, ClrGray, ClrDarkGray, ClrGray, ClrWhite, g_psFontCmss14,
        100, 50, NUM_KEYBOARD_US_ENGLISH, g_psKeyboardUSEnglish, KeyPress);
```

## 10.2 Definitions

### Data Structures

- [tKeyboard](#)
- [tKeyboardWidget](#)
- [tKeyImage](#)
- [tKeyText](#)

### Defines

- [Keyboard](#)(sName, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32BackgroundColor, ui32FillColor, ui32PressFillColor, ui32OutlineColor, ui32TextColor, psFont, ui16AutoRepeatDelay, ui16AutoRepeatRate, ui32NumKeyboards, psKeyboards, pfnOnEvent)
- [KEYBOARD\\_STYLE\\_AUTO\\_REPEAT](#)
- [KEYBOARD\\_STYLE\\_BG](#)
- [KEYBOARD\\_STYLE\\_FILL](#)
- [KEYBOARD\\_STYLE\\_IMG](#)
- [KEYBOARD\\_STYLE\\_OUTLINE](#)
- [KEYBOARD\\_STYLE\\_PRESS\\_NOTIFY](#)
- [KEYBOARD\\_STYLE\\_RELEASE\\_NOTIFY](#)
- [KEYBOARD\\_STYLE\\_TEXT](#)
- [KEYBOARD\\_STYLE\\_TEXT\\_OPAQUE](#)
- [KeyboardAutoRepeatDelaySet](#)(psWidget, ui16Delay)
- [KeyboardAutoRepeatOff](#)(psWidget)
- [KeyboardAutoRepeatOn](#)(psWidget)
- [KeyboardAutoRepeatRateSet](#)(psWidget, ui16Rate)
- [KeyboardCallbackSet](#)(psWidget, pfnOnEventFn)
- [KeyboardFillColorPressedSet](#)(psWidget, ui32Color)
- [KeyboardFillColorSet](#)(psWidget, ui32Color)
- [KeyboardFillOff](#)(psWidget)
- [KeyboardFillOn](#)(psWidget)
- [KeyboardFontSet](#)(psWidget, pFnt)
- [KeyboardOutlineColorSet](#)(psWidget, ui32Color)

- [KeyboardOutlineOff](#)(psWidget)
- [KeyboardOutlineOn](#)(psWidget)
- [KeyboardStruct](#)(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32BackgroundColor, ui32FillColor, ui32PressFillColor, ui32OutlineColor, ui32TextColor, psFont, ui16AutoRepeatDelay, ui16AutoRepeatRate, ui32NumKeyboards, psKeyboards, pfnOnEvent)
- [KeyboardTextColorSet](#)(psWidget, ui32Color)
- [KeyboardTextOpaqueOff](#)(psWidget)
- [KeyboardTextOpaqueOn](#)(psWidget)
- [NUM\\_KEYBOARD\\_US\\_ENGLISH](#)
- [UNICODE\\_BACKSPACE](#)
- [UNICODE\\_CUSTOM\\_KBD](#)
- [UNICODE\\_CUSTOM\\_LOWCASE](#)
- [UNICODE\\_CUSTOM\\_MODE\\_TOG](#)
- [UNICODE\\_CUSTOM\\_NUMERIC](#)
- [UNICODE\\_CUSTOM\\_SHIFT](#)
- [UNICODE\\_CUSTOM\\_UPCASE](#)
- [UNICODE\\_RETURN](#)

## Functions

- void [KeyboardInit](#) (tKeyboardWidget \*psWidget, const tDisplay \*psDisplay, int32\_t i32X, int32\_t i32Y, int32\_t i32Width, int32\_t i32Height)
- int32\_t [KeyboardMsgProc](#) (tWidget \*psWidget, uint32\_t ui32Msg, uint32\_t ui32Param1, uint32\_t ui32Param2)

### 10.2.1 Detailed Description

The code for this widget is contained in `grrlib/keyboard.c`, with `grrlib/keyboard.h` containing the API declarations for use by applications.

### 10.2.2 Data Structure Documentation

#### 10.2.2.1 tKeyboard

**Definition:**

```
typedef struct
{
    uint32_t ui32Code;
    uint16_t ui16NumKeys;
    uint16_t ui16Flags;
    const tKeyImage *psKeysImage;
    const tKeyText *psKeysText;
    tKeyboard::@0 uKeys;
}
tKeyboard
```

**Members:**

**ui32Code** This value holds the identifier for this keyboard.

**ui16NumKeys** This value holds the total number of keys for this keyboard entry.

**ui16Flags** This value holds the static flag entries for this keyboard entry.

**psKeysImage**

**psKeysText**

**uKeys** This union holds either the text based keys or image based keys for this keyboard.

**Description:**

This structure holds a single keyboard entry. Keyboards are typically made up of an array of these structures.

### 10.2.2.2 tKeyboardWidget

**Definition:**

```
typedef struct
{
    tWidget sBase;
    uint32_t ui32Style;
    uint32_t ui32BackgroundColor;
    uint32_t ui32FillColor;
    uint32_t ui32PressFillColor;
    uint32_t ui32OutlineColor;
    uint32_t ui32TextColor;
    const tFont *psFont;
    uint16_t ui16AutoRepeatDelay;
    uint16_t ui16AutoRepeatRate;
    uint32_t ui32AutoRepeatCount;
    uint32_t ui32Active;
    uint32_t ui32NumKeyboards;
    const tKeyboard *psKeyboards;
    void (*pfnOnEvent)(tWidget *psWidget,
                      uint32_t ui32Key,
                      uint32_t ui32Event);
    uint32_t ui32KeyPressed;
    uint32_t ui32Flags;
}
tKeyboardWidget
```

**Members:**

**sBase** The generic widget information.

**ui32Style** The style for this widget. This is a set of flags defined by KEYBOARD\_STYLE\_XXX.

**ui32BackgroundColor** The 24-bit RGB color used to fill background of the on-screen keyboard if KEYBOARD\_STYLE\_BG is selected.

**ui32FillColor** The 24-bit RGB color used to fill keys of the on-screen keyboard if KEYBOARD\_STYLE\_FILL is selected, and to use as the background color if KEYBOARD\_STYLE\_TEXT\_OPAQUE is selected.

**ui32PressFillColor** The 24-bit RGB color used to fill keys when pressed, if KEYBOARD\_STYLE\_FILL is selected, and to use as the background color if KEYBOARD\_STYLE\_TEXT\_OPAQUE is selected.

**ui32OutlineColor** The 24-bit RGB color used to outline the keys, if `KEYBOARD_STYLE_OUTLINE` is selected.

**ui32TextColor** The 24-bit RGB color used to draw text on the keys.

**psFont** A pointer to the font used to render the text on the keys.

**ui16AutoRepeatDelay** The number of pointer events to delay before starting to auto-repeat, if `KEYBOARD_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.

**ui16AutoRepeatRate** The number of pointer events between key presses generated by the auto-repeat function, if `KEYBOARD_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.

**ui32AutoRepeatCount** The number of pointer events that have occurred. This is used when `KEYBOARD_STYLE_AUTO_REPEAT` is selected to generate the auto-repeat events.

**ui32Active** The active keyboard index, which should be initialized to 0.

**ui32NumKeyboards** The total number of active keyboards in the `psKeyboards` structure member.

**psKeyboards** The array of keyboards used by the application.

**pfnOnEvent** A pointer to the function to be called when a key is pressed. This is repeatedly called when `KEYBOARD_STYLE_AUTO_REPEAT` is selected.

**ui32KeyPressed** The active key being pressed.

**ui32Flags** Internal state flags for the keyboard.

#### Description:

The structure that describes a keyboard widget.

### 10.2.2.3 tKeyImage

#### Definition:

```
typedef struct
{
    uint32_t ui32Code;
    uint16_t ui16Width;
    uint16_t ui16Height;
    uint16_t ui16XPos;
    uint16_t ui16YPos;
    const uint8_t *pui8Image;
    const uint8_t *pui8PressImage;
}
tKeyImage
```

#### Members:

**ui32Code**

**ui16Width**

**ui16Height**

**ui16XPos**

**ui16YPos**

**pui8Image** A pointer to the image to be drawn onto this key, if `KEYBOARD_STYLE_IMG` is selected.

***ui8PressImage*** A pointer to the image to be drawn onto this key when it is pressed, if KEYBOARD\_STYLE\_IMG is selected.

**Description:**

The structure to describe a image based key on the keyboard.

### 10.2.2.4 tKeyText

**Definition:**

```
typedef struct
{
    uint32_t ui32Code;
    uint16_t ui16Width;
    uint16_t ui16Height;
    uint16_t ui16XPos;
    uint16_t ui16YPos;
}
tKeyText
```

**Members:**

***ui32Code***

***ui16Width***

***ui16Height***

***ui16XPos***

***ui16YPos***

**Description:**

The structure to describe a text based key on the keyboard.

## 10.2.3 Define Documentation

### 10.2.3.1 Keyboard

Declares an initialized variable containing a keyboard widget data structure.

**Definition:**

```
#define Keyboard(sName,
                psParent,
                psNext,
                psChild,
                psDisplay,
                i32X,
                i32Y,
                i32Width,
                i32Height,
                ui32Style,
                ui32BackgroundColor,
                ui32FillColor,
                ui32PressFillColor,
                ui32OutlineColor,
```



```

    ui32TextColor,
    psFont,
    ui16AutoRepeatDelay,
    ui16AutoRepeatRate,
    ui32NumKeyboards,
    psKeyboards,
    pfnOnEvent)

```

**Parameters:**

- sName*** is the name of the variable to be declared.
- psParent*** is a pointer to the parent widget.
- psNext*** is a pointer to the sibling widget.
- psChild*** is a pointer to the first child widget.
- psDisplay*** is a pointer to the display on which to draw the keyboard.
- i32X*** is the X coordinate of the upper left corner of the keyboard.
- i32Y*** is the Y coordinate of the upper left corner of the keyboard.
- i32Width*** is the width of the keyboard.
- i32Height*** is the height of the keyboard.
- ui32Style*** is the style to be applied to the keyboard.
- ui32BackgroundColor*** is the background color for the keyboard.
- ui32FillColor*** is the color used to fill in the keys.
- ui32PressFillColor*** is the color used to fill in the keys when pressed.
- ui32OutlineColor*** is the color used to outline the keys.
- ui32TextColor*** is the color used to draw text on the keys.
- psFont*** is a pointer to the font to be used to draw text on the keys.
- ui16AutoRepeatDelay*** is the delay before starting auto-repeat.
- ui16AutoRepeatRate*** is the rate at which auto-repeat events are generated.
- ui32NumKeyboards*** is the number of keyboards in the *psKeyboards* array.
- psKeyboards*** is an array of keyboards that are displayed as a part of this keyboard.
- pfnOnEvent*** is a pointer to the function that is called when a key is pressed.

**Description:**

This macro provides an initialized keyboard 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).

*ui32Style* is the logical OR of the following:

- **PB\_STYLE\_OUTLINE** to indicate that the keys should be outlined.
- **PB\_STYLE\_FILL** to indicate that the keys should be filled.
- **PB\_STYLE\_IMG** to indicate that the keys should have an image drawn on them (using *pui8Image*).
- **PB\_STYLE\_TEXT\_OPAQUE** to indicate that the key 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 a key is released. If absent, the callback is called when a key is initially pressed.

**Returns:**

Nothing; this is not a function.

### 10.2.3.2 KEYBOARD\_STYLE\_AUTO\_REPEAT

**Definition:**

```
#define KEYBOARD_STYLE_AUTO_REPEAT
```

**Description:**

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

### 10.2.3.3 KEYBOARD\_STYLE\_BG

**Definition:**

```
#define KEYBOARD_STYLE_BG
```

**Description:**

This flag indicates that the keys should be filled.

### 10.2.3.4 KEYBOARD\_STYLE\_FILL

**Definition:**

```
#define KEYBOARD_STYLE_FILL
```

**Description:**

This flag indicates that the keys should be filled.

### 10.2.3.5 KEYBOARD\_STYLE\_IMG

**Definition:**

```
#define KEYBOARD_STYLE_IMG
```

**Description:**

This flag indicates that the keys should have an image drawn on them.

### 10.2.3.6 KEYBOARD\_STYLE\_OUTLINE

**Definition:**

```
#define KEYBOARD_STYLE_OUTLINE
```

**Description:**

This flag indicates that the keys should be outlined.

### 10.2.3.7 KEYBOARD\_STYLE\_PRESS\_NOTIFY

**Definition:**

```
#define KEYBOARD_STYLE_PRESS_NOTIFY
```

**Description:**

This flag indicates that a key is pressed.

### 10.2.3.8 KEYBOARD\_STYLE\_RELEASE\_NOTIFY

**Definition:**

```
#define KEYBOARD_STYLE_RELEASE_NOTIFY
```

**Description:**

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

### 10.2.3.9 KEYBOARD\_STYLE\_TEXT

**Definition:**

```
#define KEYBOARD_STYLE_TEXT
```

**Description:**

This flag indicates that the keys should have text drawn on them.

### 10.2.3.10 KEYBOARD\_STYLE\_TEXT\_OPAQUE

**Definition:**

```
#define KEYBOARD_STYLE_TEXT_OPAQUE
```

**Description:**

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

### 10.2.3.11 KeyboardAutoRepeatDelaySet

Sets the auto-repeat delay for a keyboard widget.

**Definition:**

```
#define KeyboardAutoRepeatDelaySet (psWidget,  
                                   ui16Delay)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

***ui16Delay*** 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 a key is pressed.

**Returns:**

None.

### 10.2.3.12 KeyboardAutoRepeatOff

Disables auto-repeat for a keyboard widget.

**Definition:**

```
#define KeyboardAutoRepeatOff (psWidget)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

**Description:**

This function disables the auto-repeat behavior of a keyboard.

**Returns:**

None.

### 10.2.3.13 KeyboardAutoRepeatOn

Enables auto-repeat for a keyboard widget.

**Definition:**

```
#define KeyboardAutoRepeatOn (psWidget)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

**Description:**

This function enables the auto-repeat behavior of a keyboard. Unpredictable behavior will occur if this is called while a key is pressed.

**Returns:**

None.

### 10.2.3.14 KeyboardAutoRepeatRateSet

Sets the auto-repeat rate for a keyboard widget.

**Definition:**

```
#define KeyboardAutoRepeatRateSet (psWidget,  
                                   ui16Rate)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

***ui16Rate*** 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 a key is pressed.

**Returns:**

None.

### 10.2.3.15 KeyboardCallbackSet

Sets the function to call when this keyboard widget is pressed.

**Definition:**

```
#define KeyboardCallbackSet (psWidget,  
                             pfnOnEventFn)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

***pfnOnEventFn*** is a pointer to the function to call.

**Description:**

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

**Returns:**

None.

### 10.2.3.16 KeyboardFillColorPressedSet

Sets the fill color of a keyboard when it is pressed.

**Definition:**

```
#define KeyboardFillColorPressedSet (psWidget,  
                                     ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to be modified.

***ui32Color*** is the 24-bit RGB color to use to fill the keys when they are pressed.

**Description:**

This function changes the color used to fill the keys on the display when a key is pressed. The display is not updated until the next paint request.

**Returns:**

None.

### 10.2.3.17 KeyboardFillColorSet

Sets the fill color of a keyboard widget.

**Definition:**

```
#define KeyboardFillColorSet (psWidget,  
                              ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to be modified.

***ui32Color*** is the 24-bit RGB color to use to fill the keys.

**Description:**

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

**Returns:**

None.

### 10.2.3.18 KeyboardFillOff

Disables filling of keys in a keyboard widget.

**Definition:**

```
#define KeyboardFillOff(psWidget)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

**Description:**

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

**Returns:**

None.

### 10.2.3.19 KeyboardFillOn

Enables filling of a keys in a keyboard widget.

**Definition:**

```
#define KeyboardFillOn(psWidget)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

**Description:**

This function enables the filling of a push key in a keyboard widget. The display is not updated until the next paint request.

**Returns:**

None.

### 10.2.3.20 KeyboardFontSet

Sets the font for a keyboard widget.

**Definition:**

```
#define KeyboardFontSet(psWidget,  
                        pFnt)
```

**Parameters:**

*psWidget* is a pointer to the keyboard widget to modify.  
*pFnt* is a pointer to the font to use to draw text on the keyboard.

**Description:**

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

**Returns:**

None.

### 10.2.3.21 KeyboardOutlineColorSet

Sets the outline color for keys in a keyboard widget.

**Definition:**

```
#define KeyboardOutlineColorSet (psWidget,  
                                ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the keyboard widget to be modified.  
*ui32Color* is the 24-bit RGB color to use to outline the keys.

**Description:**

This function changes the color used to outline the keys in a keyboard on the display. The display is not updated until the next paint request.

**Returns:**

None.

### 10.2.3.22 KeyboardOutlineOff

Disables outlining of keys in a keyboard widget.

**Definition:**

```
#define KeyboardOutlineOff (psWidget)
```

**Parameters:**

*psWidget* is a pointer to the keyboard widget to modify.

**Description:**

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

**Returns:**

None.

### 10.2.3.23 KeyboardOutlineOn

Enables outlining of keys in a keyboard widget.

**Definition:**

```
#define KeyboardOutlineOn (psWidget)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to modify.

**Description:**

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

**Returns:**

None.

### 10.2.3.24 KeyboardStruct

Declares an initialized keyboard widget data structure.

**Definition:**

```
#define KeyboardStruct (psParent,  
                        psNext,  
                        psChild,  
                        psDisplay,  
                        i32X,  
                        i32Y,  
                        i32Width,  
                        i32Height,  
                        ui32Style,  
                        ui32BackgroundColor,  
                        ui32FillColor,  
                        ui32PressFillColor,  
                        ui32OutlineColor,  
                        ui32TextColor,  
                        psFont,  
                        ui16AutoRepeatDelay,  
                        ui16AutoRepeatRate,  
                        ui32NumKeyboards,  
                        psKeyboards,  
                        pfnOnEvent)
```

**Parameters:**

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

***psDisplay*** is a pointer to the display on which to draw the keyboard.

***i32X*** is the X coordinate of the upper left corner of the keyboard.

***i32Y*** is the Y coordinate of the upper left corner of the keyboard.

***i32Width*** is the width of the keyboard.



***ui32Height*** is the height of the keyboard.  
***ui32Style*** is the style to be applied to the keyboard.  
***ui32BackgroundColor*** is the background color for the keyboard.  
***ui32FillColor*** is the color used to fill in the keyboard.  
***ui32PressFillColor*** is the color used to fill in the keyboard when a key is pressed.  
***ui32OutlineColor*** is the color used to outline the keys.  
***ui32TextColor*** is the color used to draw text on the keys.  
***psFont*** is a pointer to the font used to draw text on the keys.  
***ui16AutoRepeatDelay*** is the delay before starting auto-repeat.  
***ui16AutoRepeatRate*** is the rate at which auto-repeat events are generated.  
***ui32NumKeyboards*** is the number of keyboards in the `psKeyboards` parameter.  
***psKeyboards*** is a pointer to the keyboard array to use for this keyboard.  
***pfnOnEvent*** is a pointer to the function that is called when a key is pressed.

**Description:**

This macro provides an initialized keyboard 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:

```
tKeyboardWidget g_sKeyboard = KeyboardStruct(...);
```

Or, in an array of variables:

```
tKeyboardWidget g_psKeyboards[] =
{
    KeyboardStruct(...),
    KeyboardStruct(...),
};
```

***ui32Style*** is the logical OR of the following:

- **PB\_STYLE\_OUTLINE** to indicate that the keys should be outlined.
- **PB\_STYLE\_FILL** to indicate that the keys should be filled.
- **PB\_STYLE\_IMG** to indicate that the keys should have an image drawn on them (using *pui8Image*).
- **PB\_STYLE\_TEXT\_OPAQUE** to indicate that the key 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 a key is released. If absent, the callback is called when the key is initially pressed.

**Returns:**

Nothing; this is not a function.

### 10.2.3.25 KeyboardTextColorSet

Sets the text color of keys in a keyboard widget.

**Definition:**

```
#define KeyboardTextColorSet(psWidget,
                             ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the keyboard widget to be modified.

*ui32Color* is the 24-bit RGB color to use to draw text on the keys.

**Description:**

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

**Returns:**

None.

### 10.2.3.26 KeyboardTextOpaqueOff

Disables opaque text on keys in a keyboard widget.

**Definition:**

```
#define KeyboardTextOpaqueOff(psWidget)
```

**Parameters:**

*psWidget* is a pointer to the keyboard widget to modify.

**Description:**

This function disables the use of opaque text on a keyboard. 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 key image) to show through the text.

**Returns:**

None.

### 10.2.3.27 KeyboardTextOpaqueOn

Enables opaque text on a keyboard widget.

**Definition:**

```
#define KeyboardTextOpaqueOn(psWidget)
```

**Parameters:**

*psWidget* is a pointer to the keyboard widget to modify.

**Description:**

This function enables the use of opaque text on a keyboard. 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.

### 10.2.3.28 NUM\_KEYBOARD\_US\_ENGLISH

**Definition:**

```
#define NUM_KEYBOARD_US_ENGLISH
```

**Description:**

The total number of keyboards in the `g_psKeyboardUSEnglish` array.

### 10.2.3.29 UNICODE\_BACKSPACE

**Definition:**

```
#define UNICODE_BACKSPACE
```

**Description:**

This code is used to map a backspace key onto a keyboard. This is used in the `tKey-Text.ui32Code` or `tKeyImage.ui32Code` values.

### 10.2.3.30 UNICODE\_CUSTOM\_KBD

**Definition:**

```
#define UNICODE_CUSTOM_KBD
```

**Description:**

This code is used to identify the first custom keyboard entry.

### 10.2.3.31 UNICODE\_CUSTOM\_LOWCASE

**Definition:**

```
#define UNICODE_CUSTOM_LOWCASE
```

**Description:**

This code is used to identify a keyboard as the lower-case keyboard.

### 10.2.3.32 UNICODE\_CUSTOM\_MODE\_TOG

**Definition:**

```
#define UNICODE_CUSTOM_MODE_TOG
```

**Description:**

This code is used to map a mode toggle key onto a keyboard. This value causes the keyboard to toggle between the custom entries in a keyboard. This value is used in the `tKey-Text.ui32Code` or `tKeyImage.ui32Code` values.

### 10.2.3.33 UNICODE\_CUSTOM\_NUMERIC

**Definition:**

```
#define UNICODE_CUSTOM_NUMERIC
```

**Description:**

This code is used to identify a keyboard as the lower-case keyboard.

### 10.2.3.34 UNICODE\_CUSTOM\_SHIFT

**Definition:**

```
#define UNICODE_CUSTOM_SHIFT
```

**Description:**

This code is used to map a shift/caps-lock key onto a keyboard. This value causes the keyboard to toggle between lower-case, upper-case and caps lock modes. This value is used in the `tKeyText.ui32Code` or `tKeyImage.ui32Code` values.

### 10.2.3.35 UNICODE\_CUSTOM\_UPCASE

**Definition:**

```
#define UNICODE_CUSTOM_UPCASE
```

**Description:**

This code is used to identify a keyboard as the upper-case keyboard.

### 10.2.3.36 UNICODE\_RETURN

**Definition:**

```
#define UNICODE_RETURN
```

**Description:**

This code is used to map a return/enter key onto a keyboard. This is used in the `tKeyText.ui32Code` or `tKeyImage.ui32Code` values.

## 10.2.4 Function Documentation

### 10.2.4.1 KeyboardInit

Initializes a keyboard widget.

**Prototype:**

```
void  
KeyboardInit(tKeyboardWidget *psWidget,  
             const tDisplay *psDisplay,  
             int32_t i32X,  
             int32_t i32Y,
```

```
int32_t i32Width,  
int32_t i32Height)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget to initialize.

***psDisplay*** is a pointer to the display on which to draw the on-screen keyboard.

***i32X*** is the X coordinate of the upper left corner of the on-screen keyboard.

***i32Y*** is the Y coordinate of the upper left corner of the on-screen keyboard.

***i32Width*** is the width of the on-screen keyboard.

***i32Height*** is the height of the on-screen keyboard.

**Description:**

This function initializes the provided keyboard widget so that it is ready to be drawn when requested.

**Returns:**

None.

#### 10.2.4.2 KeyboardMsgProc

Handles messages for a rectangular keyboard widget.

**Prototype:**

```
int32_t  
KeyboardMsgProc(tWidget *psWidget,  
                uint32_t ui32Msg,  
                uint32_t ui32Param1,  
                uint32_t ui32Param2)
```

**Parameters:**

***psWidget*** is a pointer to the keyboard widget.

***ui32Msg*** is the message.

***ui32Param1*** is the first parameter to the message.

***ui32Param2*** is the second parameter to the message.

**Description:**

This function receives messages intended for this keyboard 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 Push Button Widget

Introduction .....	199
Definitions .....	199

## 11.1 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).

## 11.2 Definitions

### Data Structures

- [tPushButtonWidget](#)

## Defines

- [CircularButton](#)(sName, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32R, ui32Style, ui32FillColor, ui32PressFillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pui8PressImage, ui16AutoRepeatDelay, ui16AutoRepeatRate, pfnOnClick)
- [CircularButtonStruct](#)(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32R, ui32Style, ui32FillColor, ui32PressFillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pui8PressImage, ui16AutoRepeatDelay, ui16AutoRepeatRate, 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](#)(psWidget, ui16Delay)
- [PushButtonAutoRepeatOff](#)(psWidget)
- [PushButtonAutoRepeatOn](#)(psWidget)
- [PushButtonAutoRepeatRateSet](#)(psWidget, ui16Rate)
- [PushButtonCallbackSet](#)(psWidget, pfnOnClick)
- [PushButtonFillColorPressedSet](#)(psWidget, ui32Color)
- [PushButtonFillColorSet](#)(psWidget, ui32Color)
- [PushButtonFillOff](#)(psWidget)
- [PushButtonFillOn](#)(psWidget)
- [PushButtonFontSet](#)(psWidget, pFnt)
- [PushButtonImageOff](#)(psWidget)
- [PushButtonImageOn](#)(psWidget)
- [PushButtonImagePressedSet](#)(psWidget, plmg)
- [PushButtonImageSet](#)(psWidget, plmg)
- [PushButtonOutlineColorSet](#)(psWidget, ui32Color)
- [PushButtonOutlineOff](#)(psWidget)
- [PushButtonOutlineOn](#)(psWidget)
- [PushButtonTextColorSet](#)(psWidget, ui32Color)
- [PushButtonTextOff](#)(psWidget)
- [PushButtonTextOn](#)(psWidget)
- [PushButtonTextOpaqueOff](#)(psWidget)
- [PushButtonTextOpaqueOn](#)(psWidget)
- [PushButtonTextSet](#)(psWidget, pcTxt)
- [RectangularButton](#)(sName, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32FillColor, ui32PressFillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pui8PressImage, ui16AutoRepeatDelay, ui16AutoRepeatRate, pfnOnClick)
- [RectangularButtonStruct](#)(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui32Style, ui32FillColor, ui32PressFillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pui8PressImage, ui16AutoRepeatDelay, ui16AutoRepeatRate, pfnOnClick)



## Functions

- void `CircularButtonInit` (`tPushButtonWidget *psWidget`, const `tDisplay *psDisplay`, `int32_t i32X`, `int32_t i32Y`, `int32_t i32R`)
- `int32_t CircularButtonMsgProc` (`tWidget *psWidget`, `uint32_t ui32Msg`, `uint32_t ui32Param1`, `uint32_t ui32Param2`)
- void `RectangularButtonInit` (`tPushButtonWidget *psWidget`, const `tDisplay *psDisplay`, `int32_t i32X`, `int32_t i32Y`, `int32_t i32Width`, `int32_t i32Height`)
- `int32_t RectangularButtonMsgProc` (`tWidget *psWidget`, `uint32_t ui32Msg`, `uint32_t ui32Param1`, `uint32_t ui32Param2`)

### 11.2.1 Detailed Description

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

### 11.2.2 Data Structure Documentation

#### 11.2.2.1 `tPushButtonWidget`

**Definition:**

```
typedef struct
{
    tWidget sBase;
    uint32_t ui32Style;
    uint32_t ui32FillColor;
    uint32_t ui32PressFillColor;
    uint32_t ui32OutlineColor;
    uint32_t ui32TextColor;
    const tFont *psFont;
    const char *pcText;
    const uint8_t *pui8Image;
    const uint8_t *pui8PressImage;
    uint16_t ui16AutoRepeatDelay;
    uint16_t ui16AutoRepeatRate;
    uint32_t ui32AutoRepeatCount;
    void (*pfnOnClick)(tWidget *psWidget);
}
tPushButtonWidget
```

**Members:**

***sBase*** The generic widget information.

***ui32Style*** The style for this widget. This is a set of flags defined by `PB_STYLE_XXX`.

***ui32FillColor*** 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.

***ui32PressFillColor*** 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.

- ui32OutlineColor** The 24-bit RGB color used to outline this push button, if PB\_STYLE\_OUTLINE is selected.
- ui32TextColor** The 24-bit RGB color used to draw text on this push button, if PB\_STYLE\_TEXT is selected.
- psFont** 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.
- pui8Image** A pointer to the image to be drawn onto this push button, if PB\_STYLE\_IMG is selected.
- pui8PressImage** A pointer to the image to be drawn onto this push button when it is pressed, if PB\_STYLE\_IMG is selected.
- ui16AutoRepeatDelay** 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.
- ui16AutoRepeatRate** 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.
- ui32AutoRepeatCount** 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.

## 11.2.3 Define Documentation

### 11.2.3.1 CircularButton

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

**Definition:**

```
#define CircularButton(sName,  
    psParent,  
    psNext,  
    psChild,  
    psDisplay,  
    i32X,  
    i32Y,  
    i32R,  
    ui32Style,  
    ui32FillColor,  
    ui32PressFillColor,  
    ui32OutlineColor,  
    ui32TextColor,  
    psFont,  
    pcText,  
    pui8Image,  
    pui8PressImage,
```

```

    ui16AutoRepeatDelay,
    ui16AutoRepeatRate,
    pfnOnClick)

```

**Parameters:**

- sName*** is the name of the variable to be declared.
- psParent*** is a pointer to the parent widget.
- psNext*** is a pointer to the sibling widget.
- psChild*** is a pointer to the first child widget.
- psDisplay*** is a pointer to the display on which to draw the push button.
- i32X*** is the X coordinate of the center of the push button.
- i32Y*** is the Y coordinate of the center of the push button.
- i32R*** is the radius of the push button.
- ui32Style*** is the style to be applied to the push button.
- ui32FillColor*** is the color used to fill in the push button.
- ui32PressFillColor*** is the color used to fill in the push button when it is pressed.
- ui32OutlineColor*** is the color used to outline the push button.
- ui32TextColor*** is the color used to draw text on the push button.
- psFont*** 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.
- pui8Image*** is a pointer to the image to draw on this push button.
- pui8PressImage*** is a pointer to the image to draw on this push button when it is pressed.
- ui16AutoRepeatDelay*** is the delay before starting auto-repeat.
- ui16AutoRepeatRate*** 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).

*ui32Style* 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 *psFont* and *pcText*).
- **PB\_STYLE\_IMG** to indicate that the push button should have an image drawn on it (using *pui8Image*).
- **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.

### 11.2.3.2 CircularButtonStruct

Declares an initialized circular push button widget data structure.

**Definition:**

```
#define CircularButtonStruct (psParent,  
                             psNext,  
                             psChild,  
                             psDisplay,  
                             i32X,  
                             i32Y,  
                             i32R,  
                             ui32Style,  
                             ui32FillColor,  
                             ui32PressFillColor,  
                             ui32OutlineColor,  
                             ui32TextColor,  
                             psFont,  
                             pcText,  
                             pui8Image,  
                             pui8PressImage,  
                             ui16AutoRepeatDelay,  
                             ui16AutoRepeatRate,  
                             pfnOnClick)
```

**Parameters:**

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

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

***i32X*** is the X coordinate of the center of the push button.

***i32Y*** is the Y coordinate of the center of the push button.

***i32R*** is the radius of the push button.

***ui32Style*** is the style to be applied to the push button.

***ui32FillColor*** is the color used to fill in the push button.

***ui32PressFillColor*** is the color used to fill in the push button when it is pressed.

***ui32OutlineColor*** is the color used to outline the push button.

***ui32TextColor*** is the color used to draw text on the push button.

***psFont*** 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.

***pui8Image*** is a pointer to the image to draw on this push button.

***pui8PressImage*** is a pointer to the image to draw on this push button when it is pressed.

***ui16AutoRepeatDelay*** is the delay before starting auto-repeat.

***ui16AutoRepeatRate*** 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(...)
};
```

*ui32Style* 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 *psFont* and *pcText*).
- **PB\_STYLE\_IMG** to indicate that the push button should have an image drawn on it (using *pui8Image*).
- **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.

### 11.2.3.3 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.

### 11.2.3.4 PB\_STYLE\_FILL

**Definition:**

```
#define PB_STYLE_FILL
```

**Description:**

This flag indicates that the push button should be filled.

### 11.2.3.5 PB\_STYLE\_IMG

**Definition:**

```
#define PB_STYLE_IMG
```

**Description:**

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

### 11.2.3.6 PB\_STYLE\_OUTLINE

**Definition:**

```
#define PB_STYLE_OUTLINE
```

**Description:**

This flag indicates that the push button should be outlined.

### 11.2.3.7 PB\_STYLE\_PRESSED

**Definition:**

```
#define PB_STYLE_PRESSED
```

**Description:**

This flag indicates that the push button is pressed.

### 11.2.3.8 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.

### 11.2.3.9 PB\_STYLE\_TEXT

**Definition:**

```
#define PB_STYLE_TEXT
```

**Description:**

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

### 11.2.3.10 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).

### 11.2.3.11 PushButtonAutoRepeatDelaySet

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

**Definition:**

```
#define PushButtonAutoRepeatDelaySet (psWidget,  
                                     ui16Delay)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to modify.

***ui16Delay*** 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.

### 11.2.3.12 PushButtonAutoRepeatOff

Disables auto-repeat for a push button widget.

**Definition:**

```
#define PushButtonAutoRepeatOff (psWidget)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to modify.

**Description:**

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

**Returns:**

None.

### 11.2.3.13 PushButtonAutoRepeatOn

Enables auto-repeat for a push button widget.

**Definition:**

```
#define PushButtonAutoRepeatOn (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 11.2.3.14 PushButtonAutoRepeatRateSet

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

**Definition:**

```
#define PushButtonAutoRepeatRateSet (psWidget,  
                                     ui16Rate)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to modify.

***ui16Rate*** 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.

### 11.2.3.15 PushButtonCallbackSet

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

**Definition:**

```
#define PushButtonCallbackSet (psWidget,  
                               pfnOnClick)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to modify.

***pfnOnClick*** 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.

### 11.2.3.16 PushButtonFillColorPressedSet

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

**Definition:**

```
#define PushButtonFillColorPressedSet (psWidget,  
                                       ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to be modified.

***ui32Color*** 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.

### 11.2.3.17 PushButtonFillColorSet

Sets the fill color of a push button widget.

**Definition:**

```
#define PushButtonFillColorSet (psWidget,  
                               ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to be modified.

***ui32Color*** 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.

### 11.2.3.18 PushButtonFillOff

Disables filling of a push button widget.

**Definition:**

```
#define PushButtonFillOff (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 11.2.3.19 PushButtonFillOn

Enables filling of a push button widget.

**Definition:**

```
#define PushButtonFillOn (psWidget)
```

**Parameters:**

*psWidget* 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.

### 11.2.3.20 PushButtonFontSet

Sets the font for a push button widget.

**Definition:**

```
#define PushButtonFontSet (psWidget,  
                          pFnt)
```

**Parameters:**

*psWidget* 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.

### 11.2.3.21 PushButtonImageOff

Disables the image on a push button widget.

**Definition:**

```
#define PushButtonImageOff (psWidget)
```

**Parameters:**

*psWidget* 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.

### 11.2.3.22 PushButtonImageOn

Enables the image on a push button widget.

**Definition:**

```
#define PushButtonImageOn (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 11.2.3.23 PushButtonImagePressedSet

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

**Definition:**

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

**Parameters:**

***psWidget*** 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.

### 11.2.3.24 PushButtonImageSet

Changes the image drawn on a push button widget.

**Definition:**

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

**Parameters:**

***psWidget*** 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.

### 11.2.3.25 PushButtonOutlineColorSet

Sets the outline color of a push button widget.

**Definition:**

```
#define PushButtonOutlineColorSet (psWidget,  
                                ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to be modified.  
***ui32Color*** 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.

### 11.2.3.26 PushButtonOutlineOff

Disables outlining of a push button widget.

**Definition:**

```
#define PushButtonOutlineOff (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 11.2.3.27 PushButtonOutlineOn

Enables outlining of a push button widget.

**Definition:**

```
#define PushButtonOutlineOn (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 11.2.3.28 PushButtonTextColorSet

Sets the text color of a push button widget.

**Definition:**

```
#define PushButtonTextColorSet (psWidget,  
                               ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the push button widget to be modified.

*ui32Color* 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.

### 11.2.3.29 PushButtonTextOff

Disables the text on a push button widget.

**Definition:**

```
#define PushButtonTextOff (psWidget)
```

**Parameters:**

*psWidget* 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.

### 11.2.3.30 PushButtonTextOn

Enables the text on a push button widget.

**Definition:**

```
#define PushButtonTextOn (psWidget)
```

**Parameters:**

*psWidget* 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.

### 11.2.3.31 PushButtonTextOpaqueOff

Disables opaque text on a push button widget.

**Definition:**

```
#define PushButtonTextOpaqueOff (psWidget)
```

**Parameters:**

*psWidget* 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.

### 11.2.3.32 PushButtonTextOpaqueOn

Enables opaque text on a push button widget.

**Definition:**

```
#define PushButtonTextOpaqueOn (psWidget)
```

**Parameters:**

*psWidget* 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.

### 11.2.3.33 PushButtonTextSet

Changes the text drawn on a push button widget.

**Definition:**

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

**Parameters:**

***psWidget*** 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.

### 11.2.3.34 RectangularButton

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

**Definition:**

```
#define RectangularButton (sName,  
                          psParent,  
                          psNext,  
                          psChild,  
                          psDisplay,  
                          i32X,  
                          i32Y,  
                          i32Width,  
                          i32Height,  
                          ui32Style,  
                          ui32FillColor,  
                          ui32PressFillColor,  
                          ui32OutlineColor,  
                          ui32TextColor,  
                          psFont,  
                          pcText,  
                          pui8Image,  
                          pui8PressImage,  
                          ui16AutoRepeatDelay,  
                          ui16AutoRepeatRate,  
                          pfnOnClick)
```

**Parameters:**

***sName*** is the name of the variable to be declared.

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

**psDisplay** is a pointer to the display on which to draw the push button.  
**i32X** is the X coordinate of the upper left corner of the push button.  
**i32Y** is the Y coordinate of the upper left corner of the push button.  
**i32Width** is the width of the push button.  
**i32Height** is the height of the push button.  
**ui32Style** is the style to be applied to the push button.  
**ui32FillColor** is the color used to fill in the push button.  
**ui32PressFillColor** is the color used to fill in the push button when it is pressed.  
**ui32OutlineColor** is the color used to outline the push button.  
**ui32TextColor** is the color used to draw text on the push button.  
**psFont** 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.  
**pui8Image** is a pointer to the image to draw on this push button.  
**pui8PressImage** is a pointer to the image to draw on this push button when it is pressed.  
**ui16AutoRepeatDelay** is the delay before starting auto-repeat.  
**ui16AutoRepeatRate** 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).

*ui32Style* 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 *psFont* and *pcText*).
- **PB\_STYLE\_IMG** to indicate that the push button should have an image drawn on it (using *pui8Image*).
- **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.

### 11.2.3.35 RectangularButtonStruct

Declares an initialized rectangular push button widget data structure.

**Definition:**

```
#define RectangularButtonStruct (psParent,  
                                psNext,  
                                psChild,  
                                psDisplay,
```



```
i32X,  
i32Y,  
i32Width,  
i32Height,  
ui32Style,  
ui32FillColor,  
ui32PressFillColor,  
ui32OutlineColor,  
ui32TextColor,  
psFont,  
pcText,  
pui8Image,  
pui8PressImage,  
ui16AutoRepeatDelay,  
ui16AutoRepeatRate,  
pfnOnClick)
```

**Parameters:**

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

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

***i32X*** is the X coordinate of the upper left corner of the push button.

***i32Y*** is the Y coordinate of the upper left corner of the push button.

***i32Width*** is the width of the push button.

***i32Height*** is the height of the push button.

***ui32Style*** is the style to be applied to the push button.

***ui32FillColor*** is the color used to fill in the push button.

***ui32PressFillColor*** is the color used to fill in the push button when it is pressed.

***ui32OutlineColor*** is the color used to outline the push button.

***ui32TextColor*** is the color used to draw text on the push button.

***psFont*** 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.

***pui8Image*** is a pointer to the image to draw on this push button.

***pui8PressImage*** is a pointer to the image to draw on this push button when it is pressed.

***ui16AutoRepeatDelay*** is the delay before starting auto-repeat.

***ui16AutoRepeatRate*** 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(...)  
};
```

*ui32Style* 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 *psFont* and *pcText*).
- **PB\_STYLE\_IMG** to indicate that the push button should have an image drawn on it (using *pui8Image*).
- **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.

## 11.2.4 Function Documentation

### 11.2.4.1 CircularButtonInit

Initializes a circular push button widget.

**Prototype:**

```
void  
CircularButtonInit (tPushButtonWidget *psWidget,  
                   const tDisplay *psDisplay,  
                   int32_t i32X,  
                   int32_t i32Y,  
                   int32_t i32R)
```

**Parameters:**

***psWidget*** is a pointer to the push button widget to initialize.

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

***i32X*** is the X coordinate of the upper left corner of the push button.

***i32Y*** is the Y coordinate of the upper left corner of the push button.

***i32R*** 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.

### 11.2.4.2 CircularButtonMsgProc

Handles messages for a circular push button widget.

**Prototype:**

```
int32_t  
CircularButtonMsgProc (tWidget *psWidget,  
                      uint32_t ui32Msg,  
                      uint32_t ui32Param1,  
                      uint32_t ui32Param2)
```

**Parameters:**

**psWidget** is a pointer to the push button widget.

**ui32Msg** is the message.

**ui32Param1** is the first parameter to the message.

**ui32Param2** 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.

### 11.2.4.3 RectangularButtonInit

Initializes a rectangular push button widget.

**Prototype:**

```
void  
RectangularButtonInit (tPushButtonWidget *psWidget,  
                      const tDisplay *psDisplay,  
                      int32_t i32X,  
                      int32_t i32Y,  
                      int32_t i32Width,  
                      int32_t i32Height)
```

**Parameters:**

**psWidget** is a pointer to the push button widget to initialize.

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

**i32X** is the X coordinate of the upper left corner of the push button.

**i32Y** is the Y coordinate of the upper left corner of the push button.

**i32Width** is the width of the push button.

**i32Height** 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.

#### 11.2.4.4 RectangularButtonMsgProc

Handles messages for a rectangular push button widget.

**Prototype:**

```
int32_t  
RectangularButtonMsgProc (tWidget *psWidget,  
                          uint32_t ui32Msg,  
                          uint32_t ui32Param1,  
                          uint32_t ui32Param2)
```

**Parameters:**

**psWidget** is a pointer to the push button widget.

**ui32Msg** is the message.

**ui32Param1** is the first parameter to the message.

**ui32Param2** 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.

## 12 Radio Button Widget

Introduction .....	221
Definitions .....	222

### 12.1 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.

## 12.2 Definitions

### Data Structures

- [tRadioButtonWidget](#)

### Defines

- [RadioButton](#)(sName, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui16Style, ui16CircleSize, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pfnOnChange)
- [RadioButtonCallbackSet](#)(psWidget, pfnOnChg)
- [RadioButtonCircleSizeSet](#)(psWidget, ui16Size)
- [RadioButtonFillColorSet](#)(psWidget, ui32Color)
- [RadioButtonFillOff](#)(psWidget)
- [RadioButtonFillOn](#)(psWidget)
- [RadioButtonFontSet](#)(psWidget, pFnt)
- [RadioButtonImageOff](#)(psWidget)
- [RadioButtonImageOn](#)(psWidget)
- [RadioButtonImageSet](#)(psWidget, plmg)
- [RadioButtonOutlineColorSet](#)(psWidget, ui32Color)
- [RadioButtonOutlineOff](#)(psWidget)
- [RadioButtonOutlineOn](#)(psWidget)
- [RadioButtonStruct](#)(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, ui16Style, ui16CircleSize, ui32FillColor, ui32OutlineColor, ui32TextColor, psFont, pcText, pui8Image, pfnOnChange)
- [RadioButtonTextColorSet](#)(psWidget, ui32Color)
- [RadioButtonTextOff](#)(psWidget)
- [RadioButtonTextOn](#)(psWidget)
- [RadioButtonTextOpaqueOff](#)(psWidget)
- [RadioButtonTextOpaqueOn](#)(psWidget)
- [RadioButtonTextSet](#)(psWidget, pcTxt)
- [RB\\_STYLE\\_FILL](#)
- [RB\\_STYLE\\_IMG](#)
- [RB\\_STYLE\\_OUTLINE](#)
- [RB\\_STYLE\\_SELECTED](#)
- [RB\\_STYLE\\_TEXT](#)
- [RB\\_STYLE\\_TEXT\\_OPAQUE](#)

### Functions

- void [RadioButtonInit](#) (tRadioButtonWidget \*psWidget, const tDisplay \*psDisplay, int32\_t i32X, int32\_t i32Y, int32\_t i32Width, int32\_t i32Height)
- int32\_t [RadioButtonMsgProc](#) (tWidget \*psWidget, uint32\_t ui32Msg, uint32\_t ui32Param1, uint32\_t ui32Param2)

## 12.2.1 Detailed Description

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

## 12.2.2 Data Structure Documentation

### 12.2.2.1 tRadioButtonWidget

#### Definition:

```
typedef struct
{
    tWidget sBase;
    uint16_t ui16Style;
    uint16_t ui16CircleSize;
    uint32_t ui32FillColor;
    uint32_t ui32OutlineColor;
    uint32_t ui32TextColor;
    const tFont *psFont;
    const char *pcText;
    const uint8_t *pui8Image;
    void (*pfnOnChange)(tWidget *psWidget,
                       uint32_t bSelected);
}
tRadioButtonWidget
```

#### Members:

**sBase** The generic widget information.

**ui16Style** The style for this radio button. This is a set of flags defined by `RB_STYLE_XXX`.

**ui16CircleSize** 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).

**ui32FillColor** 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.

**ui32OutlineColor** The 24-bit RGB color used to outline this radio button, if `RB_STYLE_OUTLINE` is selected.

**ui32TextColor** The 24-bit RGB color used to draw text on this radio button, if `RB_STYLE_TEXT` is selected.

**psFont** 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.

**pui8Image** 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.

## 12.2.3 Define Documentation

### 12.2.3.1 RadioButton

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

**Definition:**

```
#define RadioButton(sName,  
                  psParent,  
                  psNext,  
                  psChild,  
                  psDisplay,  
                  i32X,  
                  i32Y,  
                  i32Width,  
                  i32Height,  
                  ui16Style,  
                  ui16CircleSize,  
                  ui32FillColor,  
                  ui32OutlineColor,  
                  ui32TextColor,  
                  psFont,  
                  pcText,  
                  pui8Image,  
                  pfnOnChange)
```

**Parameters:**

***sName*** is the name of the variable to be declared.

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

***psDisplay*** is a pointer to the display on which to draw the radio button.

***i32X*** is the X coordinate of the upper left corner of the radio button.

***i32Y*** is the Y coordinate of the upper left corner of the radio button.

***i32Width*** is the width of the radio button.

***i32Height*** is the height of the radio button.

***ui16Style*** is the style to be applied to this radio button.

***ui16CircleSize*** is the size of the circle that is filled.

***ui32FillColor*** is the color used to fill in the radio button.

***ui32OutlineColor*** is the color used to outline the radio button.

***ui32TextColor*** is the color used to draw text on the radio button.

***psFont*** 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.

***pui8Image*** 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).



*ui16Style* 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 *psFont* and *pcText*).
- **RB\_STYLE\_IMG** to indicate that the radio button should have an image drawn on it (using *pui8Image*).
- **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.

### 12.2.3.2 RadioButtonCallbackSet

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

**Definition:**

```
#define RadioButtonCallbackSet (psWidget,  
                               pfnOnChg)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.3 RadioButtonCircleSizeSet

Sets size of the circle to be filled.

**Definition:**

```
#define RadioButtonCircleSizeSet (psWidget,  
                                  ui16Size)
```

**Parameters:**

***psWidget*** is a pointer to the radio button widget to modify.  
***ui16Size*** 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.

#### 12.2.3.4 RadioButtonFillColorSet

Sets the fill color of a radio button widget.

**Definition:**

```
#define RadioButtonFillColorSet (psWidget,  
                                ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the radio button widget to be modified.  
***ui32Color*** 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.

#### 12.2.3.5 RadioButtonFillOff

Disables filling of a radio button widget.

**Definition:**

```
#define RadioButtonFillOff (psWidget)
```

**Parameters:**

***psWidget*** 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.

#### 12.2.3.6 RadioButtonFillOn

Enables filling of a radio button widget.

**Definition:**

```
#define RadioButtonFillOn (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.7 RadioButtonFontSet

Sets the font for a radio button widget.

**Definition:**

```
#define RadioButtonFontSet (psWidget,  
                           pFnt)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.8 RadioButtonImageOff

Disables the image on a radio button widget.

**Definition:**

```
#define RadioButtonImageOff (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.9 RadioButtonImageOn

Enables the image on a radio button widget.

**Definition:**

```
#define RadioButtonImageOn (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.10 RadioButtonImageSet

Changes the image drawn on a radio button widget.

**Definition:**

```
#define RadioButtonImageSet (psWidget,  
                             pImg)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.11 RadioButtonOutlineColorSet

Sets the outline color of a radio button widget.

**Definition:**

```
#define RadioButtonOutlineColorSet (psWidget,  
                                    ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the radio button widget to be modified.

***ui32Color*** 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.

### 12.2.3.12 RadioButtonOutlineOff

Disables outlining of a radio button widget.

**Definition:**

```
#define RadioButtonOutlineOff (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.13 RadioButtonOutlineOn

Enables outlining of a radio button widget.

**Definition:**  

```
#define RadioButtonOutlineOn(psWidget)
```

**Parameters:**  
***psWidget*** 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.

### 12.2.3.14 RadioButtonStruct

Declares an initialized radio button widget data structure.

**Definition:**  

```
#define RadioButtonStruct(psParent,  
                        psNext,  
                        psChild,  
                        psDisplay,  
                        i32X,  
                        i32Y,  
                        i32Width,  
                        i32Height,  
                        ui16Style,  
                        ui16CircleSize,  
                        ui32FillColor,  
                        ui32OutlineColor,  
                        ui32TextColor,  
                        psFont,  
                        pcText,  
                        pui8Image,  
                        pfnOnChange)
```

**Parameters:**  
***psParent*** is a pointer to the parent widget.  
***psNext*** is a pointer to the sibling widget.  
***psChild*** is a pointer to the first child widget.  
***psDisplay*** is a pointer to the display on which to draw the radio button.  
***i32X*** is the X coordinate of the upper left corner of the radio button.  
***i32Y*** is the Y coordinate of the upper left corner of the radio button.

***i32Width*** is the width of the radio button.  
***i32Height*** is the height of the radio button.  
***ui16Style*** is the style to be applied to this radio button.  
***ui16CircleSize*** is the size of the circle that is filled.  
***ui32FillColor*** is the color used to fill in the radio button.  
***ui32OutlineColor*** is the color used to outline the radio button.  
***ui32TextColor*** is the color used to draw text on the radio button.  
***psFont*** 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.  
***pui8Image*** 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_s6RadioButton = RadioButtonStruct(...);
```

Or, in an array of variables:

```
tRadioButtonWidget g_psRadioButtons[] =  
{  
    RadioButtonStruct(...),  
    RadioButtonStruct(...)  
};
```

***ui16Style*** 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 *psFont* and *pcText*).
- **RB\_STYLE\_IMG** to indicate that the radio button should have an image drawn on it (using *pui8Image*).
- **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.

### 12.2.3.15 RadioButtonTextColorSet

Sets the text color of a radio button widget.

**Definition:**

```
#define RadioButtonTextColorSet (psWidget,  
                                ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the radio button widget to be modified.

***ui32Color*** 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.

### 12.2.3.16 RadioButtonTextOff

Disables the text on a radio button widget.

**Definition:**

```
#define RadioButtonTextOff(psWidget)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.17 RadioButtonTextOn

Enables the text on a radio button widget.

**Definition:**

```
#define RadioButtonTextOn(psWidget)
```

**Parameters:**

***psWidget*** 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.

### 12.2.3.18 RadioButtonTextOpaqueOff

Disables opaque text on a radio button widget.

**Definition:**

```
#define RadioButtonTextOpaqueOff(psWidget)
```

**Parameters:**

*psWidget* 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.

### 12.2.3.19 RadioButtonTextOpaqueOn

Enables opaque text on a radio button widget.

**Definition:**

```
#define RadioButtonTextOpaqueOn (psWidget)
```

**Parameters:**

*psWidget* 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.

### 12.2.3.20 RadioButtonTextSet

Changes the text drawn on a radio button widget.

**Definition:**

```
#define RadioButtonTextSet (psWidget,  
                           pcTxt)
```

**Parameters:**

*psWidget* 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.



### 12.2.3.21 RB\_STYLE\_FILL

**Definition:**

```
#define RB_STYLE_FILL
```

**Description:**

This flag indicates that the radio button should be filled.

### 12.2.3.22 RB\_STYLE\_IMG

**Definition:**

```
#define RB_STYLE_IMG
```

**Description:**

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

### 12.2.3.23 RB\_STYLE\_OUTLINE

**Definition:**

```
#define RB_STYLE_OUTLINE
```

**Description:**

This flag indicates that the radio button should be outlined.

### 12.2.3.24 RB\_STYLE\_SELECTED

**Definition:**

```
#define RB_STYLE_SELECTED
```

**Description:**

This flag indicates that the radio button is selected.

### 12.2.3.25 RB\_STYLE\_TEXT

**Definition:**

```
#define RB_STYLE_TEXT
```

**Description:**

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

### 12.2.3.26 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).

## 12.2.4 Function Documentation

### 12.2.4.1 RadioButtonInit

Initializes a radio button widget.

**Prototype:**

```
void  
RadioButtonInit (tRadioButtonWidget *psWidget,  
                const tDisplay *psDisplay,  
                int32_t i32X,  
                int32_t i32Y,  
                int32_t i32Width,  
                int32_t i32Height)
```

**Parameters:**

***psWidget*** is a pointer to the radio button widget to initialize.

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

***i32X*** is the X coordinate of the upper left corner of the radio button.

***i32Y*** is the Y coordinate of the upper left corner of the radio button.

***i32Width*** is the width of the radio button.

***i32Height*** is the height of the radio button.

**Description:**

This function initializes the provided radio button widget.

**Returns:**

None.

### 12.2.4.2 RadioButtonMsgProc

Handles messages for a radio button widget.

**Prototype:**

```
int32_t  
RadioButtonMsgProc (tWidget *psWidget,  
                   uint32_t ui32Msg,  
                   uint32_t ui32Param1,  
                   uint32_t ui32Param2)
```

**Parameters:**

***psWidget*** is a pointer to the radio button widget.

***ui32Msg*** is the message.

***ui32Param1*** is the first parameter to the message.

***ui32Param2*** 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.



## 13 Slider Widget

Introduction .....	237
Definitions .....	238

### 13.1 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.

## 13.2 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, psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, i32Min, i32Max, i32Value, ui32Style, ui32FillColor, ui32BackgroundFillColor, ui32OutlineColor, ui32TextColor, ui32BackgroundTextColor, psFont, pcText, pui8Image, pui8BackgroundImage, pfnOnChange)
- `SliderBackgroundFillOff`(psWidget)
- `SliderBackgroundFillOn`(psWidget)
- `SliderBackgroundImageOff`(psWidget)
- `SliderBackgroundImageOn`(psWidget)
- `SliderBackgroundImageSet`(psWidget, plmg)
- `SliderBackgroundTextColorSet`(psWidget, ui32Color)
- `SliderBackgroundTextOff`(psWidget)
- `SliderBackgroundTextOn`(psWidget)
- `SliderBackgroundTextOpaqueOff`(psWidget)
- `SliderBackgroundTextOpaqueOn`(psWidget)
- `SliderCallbackSet`(psWidget, pfnCallback)
- `SliderFillColorBackgroundedSet`(psWidget, ui32Color)
- `SliderFillColorSet`(psWidget, ui32Color)
- `SliderFillOff`(psWidget)
- `SliderFillOn`(psWidget)
- `SliderFontSet`(psWidget, psFnt)
- `SliderImageOff`(psWidget)
- `SliderImageOn`(psWidget)
- `SliderImageSet`(psWidget, plmg)
- `SliderLock`(psWidget)
- `SliderOutlineColorSet`(psWidget, ui32Color)
- `SliderOutlineOff`(psWidget)
- `SliderOutlineOn`(psWidget)
- `SliderRangeSet`(psWidget, i32Minimum, i32Maximum)
- `SliderStruct`(psParent, psNext, psChild, psDisplay, i32X, i32Y, i32Width, i32Height, i32Min, i32Max, i32Value, ui32Style, ui32FillColor, ui32BackgroundFillColor, ui32OutlineColor, ui32TextColor, ui32BackgroundTextColor, psFont, pcText, pui8Image, pui8BackgroundImage, pfnOnChange)
- `SliderTextColorSet`(psWidget, ui32Color)
- `SliderTextOff`(psWidget)
- `SliderTextOn`(psWidget)
- `SliderTextOpaqueOff`(psWidget)
- `SliderTextOpaqueOn`(psWidget)
- `SliderTextSet`(psWidget, pcTtxt)
- `SliderUnlock`(psWidget)
- `SliderValueSet`(psWidget, i32Val)
- `SliderVerticalSet`(psWidget, bVertical)

## Functions

- void `SliderInit` (tSliderWidget \*psWidget, const tDisplay \*psDisplay, int32\_t i32X, int32\_t i32Y, int32\_t i32Width, int32\_t i32Height)
- int32\_t `SliderMsgProc` (tWidget \*psWidget, uint32\_t ui32Msg, uint32\_t ui32Param1, uint32\_t ui32Param2)

## 13.2.1 Detailed Description

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

## 13.2.2 Data Structure Documentation

### 13.2.2.1 tSliderWidget

**Definition:**

```
typedef struct
{
    tWidget sBase;
    uint32_t ui32Style;
    uint32_t ui32FillColor;
    uint32_t ui32BackgroundFillColor;
    uint32_t ui32OutlineColor;
    uint32_t ui32TextColor;
    uint32_t ui32BackgroundTextColor;
    const tFont *psFont;
    const char *pcText;
    const uint8_t *pui8Image;
    const uint8_t *pui8BackgroundImage;
    void (*pfnOnChange)(tWidget *psWidget,
                       int32_t i32Value);

    int32_t i32Min;
    int32_t i32Max;
    int32_t i32Value;
    int16_t i16Pos;
}
tSliderWidget
```

**Members:**

**sBase** The generic widget information.

**ui32Style** The style for this widget. This is a set of flags defined by `SL_STYLE_XXX`.

**ui32FillColor** 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.

**ui32BackgroundFillColor** 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.

**ui32OutlineColor** The 24-bit RGB color used to outline this slider, if `SL_STYLE_OUTLINE` is selected.

**ui32TextColor** The 24-bit RGB color used to draw text on the "active" portion of this slider, if `SL_STYLE_TEXT` is selected.

**ui32BackgroundTextColor** The 24-bit RGB color used to draw text on the background portion of this slider, if `SL_STYLE_TEXT` is selected.

**psFont** 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.

**pui8Image** A pointer to the image to be drawn onto this slider, if `SL_STYLE_IMG` is selected.



***pui8BackgroundImage*** 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.

***i32Min*** 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.

***i32Max*** 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.

***i32Value*** The current slider value scaled according to the minimum and maximum values for the control.

***i16Pos*** This internal work variable stores the pixel position representing the current slider value.

**Description:**

The structure that describes a slider widget.

## 13.2.3 Define Documentation

### 13.2.3.1 `SL_STYLE_BACKG_FILL`

**Definition:**

```
#define SL_STYLE_BACKG_FILL
```

**Description:**

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

### 13.2.3.2 `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.

### 13.2.3.3 `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.

### 13.2.3.4 `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.

### 13.2.3.5 SL\_STYLE\_FILL

**Definition:**

```
#define SL_STYLE_FILL
```

**Description:**

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

### 13.2.3.6 SL\_STYLE\_IMG

**Definition:**

```
#define SL_STYLE_IMG
```

**Description:**

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

### 13.2.3.7 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.

### 13.2.3.8 SL\_STYLE\_OUTLINE

**Definition:**

```
#define SL_STYLE_OUTLINE
```

**Description:**

This flag indicates that the slider should be outlined.

### 13.2.3.9 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.

### 13.2.3.10 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.

### 13.2.3.11 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.

### 13.2.3.12 Slider

Declares an initialized variable containing a slider widget data structure.

**Definition:**

```
#define Slider(sName,  
              psParent,  
              psNext,  
              psChild,  
              psDisplay,  
              i32X,  
              i32Y,  
              i32Width,  
              i32Height,  
              i32Min,  
              i32Max,  
              i32Value,  
              ui32Style,  
              ui32FillColor,  
              ui32BackgroundFillColor,  
              ui32OutlineColor,  
              ui32TextColor,  
              ui32BackgroundTextColor,  
              psFont,  
              pcText,  
              pui8Image,  
              pui8BackgroundImage,  
              pfnOnChange)
```

**Parameters:**

**sName** is the name of the variable to be declared.

**psParent** is a pointer to the parent widget.

**psNext** is a pointer to the sibling widget.

**psChild** is a pointer to the first child widget.

**psDisplay** is a pointer to the display on which to draw the slider.

**i32X** is the X coordinate of the upper left corner of the slider.

**i32Y** is the Y coordinate of the upper left corner of the slider.

**i32Width** is the width of the slider.

**i32Height** is the height of the slider.

**i32Min** is the minimum value for the slider (corresponding to the left or bottom position).

**i32Max** is the maximum value for the slider (corresponding to the right or top position).

**i32Value** is the initial value of the slider. This must lie in the range defined by *i32Min* and *i32Max*.

**ui32Style** is the style to be applied to the slider.

**ui32FillColor** is the color used to fill in the slider.

**ui32BackgroundFillColor** is the color used to fill in the background area of the slider.

**ui32OutlineColor** is the color used to outline the slider.

**ui32TextColor** is the color used to draw text on the slider.

**ui32BackgroundTextColor** is the color used to draw text on the background portion of the slider.

**psFont** 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.

**pui8Image** is a pointer to the image to draw on this slider.

**pui8BackgroundImage** 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).

*ui32Style* 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 *psFont* and *pcText*).
- **SL\_STYLE\_BACKG\_TEXT** to indicate that the slider should have text drawn on its background portion (using *psFont* and *pcText*).
- **SL\_STYLE\_IMG** to indicate that the slider should have an image drawn on it (using *pui8Image*).
- **SL\_STYLE\_BACKG\_IMG** to indicate that the slider should have an image drawn on its background (using *pui8BackgroundImage*).
- **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.

### 13.2.3.13 SliderBackgroundFillOff

Disables filling of the background area of a slider widget.

**Definition:**

```
#define SliderBackgroundFillOff(psWidget)
```

**Parameters:**

***psWidget*** 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.

### 13.2.3.14 SliderBackgroundFillOn

Enables filling of the background area of a slider widget.

**Definition:**

```
#define SliderBackgroundFillOn(psWidget)
```

**Parameters:**

***psWidget*** 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.

### 13.2.3.15 SliderBackgroundImageOff

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

**Definition:**

```
#define SliderBackgroundImageOff(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.16 SliderBackgroundImageOn

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

**Definition:**

```
#define SliderBackgroundImageOn (psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.17 SliderBackgroundImageSet

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

**Definition:**

```
#define SliderBackgroundImageSet (psWidget,  
                                pImg)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.18 SliderBackgroundTextColorSet

Sets the background text color of a slider widget.

**Definition:**

```
#define SliderBackgroundTextColorSet (psWidget,  
                                     ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the slider widget to be modified.

***ui32Color*** 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.

### 13.2.3.19 SliderBackgroundTextOff

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

**Definition:**

```
#define SliderBackgroundTextOff (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 13.2.3.20 SliderBackgroundTextOn

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

**Definition:**

```
#define SliderBackgroundTextOn (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 13.2.3.21 SliderBackgroundTextOpaqueOff

Disables opaque background text on a slider widget.

**Definition:**

```
#define SliderBackgroundTextOpaqueOff (psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.22 SliderBackgroundTextOpaqueOn

Enables opaque background text on a slider widget.

**Definition:**

```
#define SliderBackgroundTextOpaqueOn (psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.23 SliderCallbackSet

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

**Definition:**

```
#define SliderCallbackSet (psWidget,  
                           pfnCallback)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.24 SliderFillColorBackgroundedSet

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

**Definition:**

```
#define SliderFillColorBackgroundedSet (psWidget,  
                                       ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the slider widget to be modified.

***ui32Color*** 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.

### 13.2.3.25 SliderFillColorSet

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

**Definition:**

```
#define SliderFillColorSet (psWidget,  
                           ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the slider widget to be modified.

***ui32Color*** 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.

### 13.2.3.26 SliderFillOff

Disables filling of the active area of a slider widget.

**Definition:**

```
#define SliderFillOff (psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.27 SliderFillOn

Enables filling of the active area of a slider widget.

**Definition:**

```
#define SliderFillOn(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.28 SliderFontSet

Sets the font for a slider widget.

**Definition:**

```
#define SliderFontSet(psWidget,  
                    psFnt)
```

**Parameters:**

*psWidget* is a pointer to the slider widget to modify.

*psFnt* 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.

### 13.2.3.29 SliderImageOff

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

**Definition:**

```
#define SliderImageOff(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.30 SliderImageOn

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

**Definition:**

```
#define SliderImageOn(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.31 SliderImageSet

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

**Definition:**

```
#define SliderImageSet(psWidget,  
                      pImg)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.32 SliderLock

Locks a slider making it ignore pointer input.

**Definition:**

```
#define SliderLock(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.33 SliderOutlineColorSet

Sets the outline color of a slider widget.

**Definition:**

```
#define SliderOutlineColorSet(psWidget,  
                             ui32Color)
```

**Parameters:**

*psWidget* is a pointer to the slider widget to be modified.  
*ui32Color* 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.

### 13.2.3.34 SliderOutlineOff

Disables outlining of a slider widget.

**Definition:**

```
#define SliderOutlineOff(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.35 SliderOutlineOn

Enables outlining of a slider widget.

**Definition:**  

```
#define SliderOutlineOn(psWidget)
```

**Parameters:**  
*psWidget* 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.

### 13.2.3.36 SliderRangeSet

Changes the value range for a slider widget.

**Definition:**  

```
#define SliderRangeSet(psWidget,  
                      i32Minimum,  
                      i32Maximum)
```

**Parameters:**  
*psWidget* is a pointer to the slider widget to be modified.  
*i32Minimum* is the minimum value that the slider will report.  
*i32Maximum* 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.

### 13.2.3.37 SliderStruct

Declares an initialized slider widget data structure.

**Definition:**

```
#define SliderStruct (psParent,  
                    psNext,  
                    psChild,  
                    psDisplay,  
                    i32X,  
                    i32Y,  
                    i32Width,  
                    i32Height,  
                    i32Min,  
                    i32Max,  
                    i32Value,  
                    ui32Style,  
                    ui32FillColor,  
                    ui32BackgroundFillColor,  
                    ui32OutlineColor,  
                    ui32TextColor,  
                    ui32BackgroundTextColor,  
                    psFont,  
                    pcText,  
                    pui8Image,  
                    pui8BackgroundImage,  
                    pfnOnChange)
```

**Parameters:**

***psParent*** is a pointer to the parent widget.

***psNext*** is a pointer to the sibling widget.

***psChild*** is a pointer to the first child widget.

***psDisplay*** is a pointer to the display on which to draw the slider.

***i32X*** is the X coordinate of the upper left corner of the slider.

***i32Y*** is the Y coordinate of the upper left corner of the slider.

***i32Width*** is the width of the slider.

***i32Height*** is the height of the slider.

***i32Min*** is the minimum value for the slider (corresponding to the left or bottom position).

***i32Max*** is the maximum value for the slider (corresponding to the right or top position).

***i32Value*** is the initial value of the slider. This must lie in the range defined by *i32Min* and *i32Max*.

***ui32Style*** is the style to be applied to the slider.

***ui32FillColor*** is the color used to fill in the slider.

***ui32BackgroundFillColor*** is the color used to fill the background area of the slider.

***ui32OutlineColor*** is the color used to outline the slider.

***ui32TextColor*** is the color used to draw text on the slider.

***ui32BackgroundTextColor*** is the color used to draw text on the background portion of the slider.

***psFont*** 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.

***pui8Image*** is a pointer to the image to draw on this slider.

***pui8BackgroundImage*** 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(...)
};
```

*ui32Style* 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 *psFont* and *pcText*).
- **SL\_STYLE\_BACKG\_TEXT** to indicate that the slider should have text drawn on its background portion (using *psFont* and *pcText*).
- **SL\_STYLE\_IMG** to indicate that the slider should have an image drawn on it (using *pui8Image*).
- **SL\_STYLE\_BACKG\_IMG** to indicate that the slider should have an image drawn on its background (using *pui8BackgroundImage*).
- **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.

### 13.2.3.38 SliderTextColorSet

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

**Definition:**

```
#define SliderTextColorSet(psWidget,
                           ui32Color)
```

**Parameters:**

***psWidget*** is a pointer to the slider widget to be modified.

*ui32Color* 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.

### 13.2.3.39 SliderTextOff

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

**Definition:**

```
#define SliderTextOff(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.40 SliderTextOn

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

**Definition:**

```
#define SliderTextOn(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.41 SliderTextOpaqueOff

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

**Definition:**

```
#define SliderTextOpaqueOff(psWidget)
```



**Parameters:**

*psWidget* 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.

### 13.2.3.42 SliderTextOpaqueOn

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

**Definition:**

```
#define SliderTextOpaqueOn(psWidget)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.43 SliderTextSet

Changes the text drawn on a slider widget.

**Definition:**

```
#define SliderTextSet(psWidget,  
                    pcTxt)
```

**Parameters:**

*psWidget* 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.

### 13.2.3.44 SliderUnlock

Unlocks a slider making it pay attention to pointer input.

**Definition:**

```
#define SliderUnlock (psWidget)
```

**Parameters:**

***psWidget*** 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.

### 13.2.3.45 SliderValueSet

Changes the minimum value for a slider widget.

**Definition:**

```
#define SliderValueSet (psWidget,  
                      i32Val)
```

**Parameters:**

***psWidget*** is a pointer to the slider widget to be modified.

***i32Val*** 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.

### 13.2.3.46 SliderVerticalSet

Sets the vertical or horizontal style for a slider widget

**Definition:**

```
#define SliderVerticalSet (psWidget,  
                          bVertical)
```

**Parameters:**

***psWidget*** 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.

## 13.2.4 Function Documentation

### 13.2.4.1 SliderInit

Initializes a slider widget.

**Prototype:**

```
void
SliderInit(tSliderWidget *psWidget,
           const tDisplay *psDisplay,
           int32_t i32X,
           int32_t i32Y,
           int32_t i32Width,
           int32_t i32Height)
```

**Parameters:**

***psWidget*** is a pointer to the slider widget to initialize.  
***psDisplay*** is a pointer to the display on which to draw the slider.  
***i32X*** is the X coordinate of the upper left corner of the slider.  
***i32Y*** is the Y coordinate of the upper left corner of the slider.  
***i32Width*** is the width of the slider.  
***i32Height*** is the height of the slider.

**Description:**

This function initializes the provided slider widget.

**Returns:**

None.

### 13.2.4.2 SliderMsgProc

Handles messages for a slider widget.

**Prototype:**

```
int32_t
SliderMsgProc(tWidget *psWidget,
             uint32_t ui32Msg,
             uint32_t ui32Param1,
             uint32_t ui32Param2)
```

**Parameters:**

*psWidget* is a pointer to the slider widget.

*ui32Msg* is the message.

*ui32Param1* is the first parameter to the message.

*ui32Param2* 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.

## 14 Utilities

Introduction .....	261
ftrasterize .....	261
lmi-button .....	265
pnmtoc .....	265
mkstringtable .....	266

### 14.1 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 make use of the graphics library, though they do make it much easier to use.

### 14.2 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 to a file that provides a `tFont`, `tFontEx` or `tFontWide` structure describing the font. The output may be written in the form of a C source file which can be linked into an application directly or as a binary file allowing the font to be stored and used from non-linear memory or a file system assuming a suitable font wrapper is used.

The application is run from the command line, and its usage is as follows:

```
ftrasterize [-a <num>] [-b] [-c <filename>] [-d] [-e <num>] [-f <name>]
            [-h] [-i] [-m] [-n] [-o <num>] [-p <num>] [-r] [-s [F]<size>]
            [-t <num>] [-u] [-v] [-w <num>] [-y] [-z <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 <name>           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. The size parameter may take one of two forms. If a decimal number, this specifies the text size in points. If the parameter is a capital "F" followed immediately by a decimal number, this selects one of the font's fixed size encodings assuming the font supports these. To determine whether a font supports fixed sizes, use the "-d" switch to display font information. The default size is 20 points 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. The switch is ignored if "-r" is specified.
- `-m` Specifies that the output font is to be monospaced. This causes the output glyphs to be generated such that the character is centered within the character cell and each character is reported as having the same width. If absent, the character widths are determined by studying the rendered bitmaps for each.
- `-n` Overrides -w and causes no character to be encoded as a space unless the source font already contains a space. The switch is ignored if "-r" is specified.
- `-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. This switch is ignored if "-c" is also specified.
- `-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 ( ). This switch is ignored if "-c" is also specified.
- `-t <num>` Used in conjunction with "-o" to allow a single contiguous block of characters at a particular position in the input font's codepage to be translated down into the 0-255 codepoint range supported by the tFont and tFontEx font types. This can be useful when generating ISO8859 font variants from Unicode fonts, for example. The parameter passed with "-t" is the codepoint in the output font at which the translated block of characters from the input will be placed. The switch is ignored if "-r" is specified.
- `-o <num>` Used in conjunction with "-t", this switch defines the codepoint in the source font representing the first character in the block of characters which is to be moved into the 256 codepoint range of the output font. Using these two switches, the output font will contain characters matching the input font with codepoints in the range from the start (either 0x20 or the value passed with "-p") to the "-t" value minus 1. Output characters with codepoints from the "-t" value and above will contain source font characters from codepoints starting at the "-o" value. The switch is ignored if "-r" is specified.

- 
- `-a <num>` Selects the font character map used when reading from the source font. The number provided is the zero-based index of the required character map. Valid values may be determined by running `frasterize` with the `"-d"` option to display information on the source font. If `'-r'` is specified and this switch is absent, the Unicode character map is used by default. If both `"-r"` and `"-a"` are absent, the Adobe Custom character map is used if it exists and the Unicode map otherwise.
- `-u` Specifies that the Unicode character mapping from the source font should be used. If absent, the Adobe Custom character map will be used by default when `"-r"` is absent. If the font does not include an Adobe Custom mapping, Unicode will be used if present. If `"-r"` is specified, the Unicode character mapping is used by default unless `"-a"` overrides this.
- `-r` Specifies that a relocatable, wide character set font using the `tFontWide` format should be generated. This format allows multiple contiguous blocks of characters to be encoded and does not have the 256 character limit imposed by the `tFontEx` format. If absent, the output format will be either `tFont` or `tFontEx` depending upon the range of characters that are to be encoded. `"-r"` is typically used to encode non-western character sets and is required even for western alphabets if the font is to be used from non-linear storage such as a file system or serial memory via a `tFontWrapper` structure.
- `-y` Specifies that the output should be a binary file. If absent, the output is a C source file suitable for building into an application. This switch is ignored if `"-r"` is not specified.
- `-c <filename>` Provides the name of the character block mapping file to be used when generating a wide character set font. This switch is ignored unless `"-r"` is specified. The file provided is a text file containing information on the blocks of source characters that are to be encoded. Each line may be whitespace, a comment starting with the `"#"` character, a single hex number indicating a single character to encode, or two hex values separated by a comma and space indicating a range of characters. When a range is specified, the range is inclusive. If the first non-comment line of the file is the string `"REMAP"`, the output font file is written using a remapped codepage with character codepoints starting at 1 and incrementing for every character encoded. The character order is determined by the order of characters defined in the character block mapping file with the first character in the first block defined in the file being assigned character code 1. This feature works alongside the remapping feature offered by the `mkstringtable` tool to allow creation of very small string tables and custom fonts and is particularly helpful when dealing with alphabets containing large numbers of possible glyphs.
- `-d` Parses the input font and displays information on its contents. If `"-d"` is specified, all other switches except `"-h"` and `"-v"` are ignored. When used without `"-v"`, font header information and properties are shown along with the total number of characters encoded by the font and the number of contiguous blocks these characters are found in. With `"-v"`, detailed information on the character blocks is also displayed.

- `-v` Enables verbose output which provides a great deal more information on the encoding progress as `ftrasterize` runs. If used with `-d` information on all character blocks encoded within the source font is output in addition to the basic font information.
- `-z <num>` Sets the output font's codepage to the supplied value. This is used to specify a custom codepage identifier when performing glyph remapping. Values should be between `CODEPAGE_CUSTOM_BASE` (0x8000) and 0xFFFF. This switch is only valid when used with `-r`.
- `-h` Displays help information on all command line switches. If specified, all other command line switches are ignored.
- `<font>` Specifies the name of the input font file to be processed.

For example, to produce a 24 point font called "test" containing ASCII characters in the range 0x20 to 0x7E 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 (the first character) is forced to be a space to ensure that a space exists in the font even though ASCII character 0x20 is not present:

```
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`. A pointer to this structure cast to a `(const tFont *)` type can be used in any graphics library API call that requires a font pointer.

To render a font containing western and Cyrillic alphabets compatible with ISO8859-5 from a Unicode font supporting the relevant glyphs, the following command line could be used. The output font will contain 20 point encodings of characters in the range 0x20-0xFF where those in the 0x20-0x9F range are taken directly from the Unicode characters in the same range (ASCII + a few additional accented characters that are undefined in ISO8859-5) and those from 0xA0-0xFF are taken from the Unicode space starting at 0x400 which contains the basic Cyrillic alphabet.

```
ftrasterize -f iso8859_5 -s 20 -p 32 -e 255 -t 128 -o 0x400 -u cyrillic.ttf
```

The output will be written to `fontiso8859_520.c` and contain a definition for `g_sFontIso8859_520`. A pointer to this structure cast to a `(const tFont *)` type can be used in any graphics library API call that requires a font pointer.

When encoding wide character sets for multiple alphabets (Roman, Arabic, Cyrillic, Hebrew, etc.) or to deal with ideograph-based writing systems (Hangul, Traditional or Simplified Chinese, Hiragana, Katakana, etc.), a character block map file is required to define which sections of the source font's codespace to encode into the destination font. The following example character map could be used to encode a font containing ASCII plus the Japanese Katakana alphabets:



```
#####
#
# katakana.txt - Unicode block definitions for ASCII and Katakana.
#
#####

# ASCII characters
0x20, 0x7E

# Katakana alphabet
0x30A0, 0x30FF
0x31F0, 0x32FF
0xFF00, 0xFFEF
```

Assuming the font “unicode.ttf” contains these glyphs and that it includes fixed size character renderings, the fifth of which uses an 8x12 character cell size, the following frasterize command line could then be used to generate a binary font file called fontkatakana8x12.bin containing this subset of characters:

```
ftrasterize -f katakana -s F4 -c katakana.txt -y -r -u unicode.ttf
```

In this case, the output file will be fontkatakana8x12.bin and it will contain a binary version of the font suitable for use from external memory (SDCard, a file system, serial flash memory, etc.) via a [tFontWrapper](#) and a suitable font wrapper module.

This application is located in `tools/ftrasterize`.

## 14.3 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`.

## 14.4 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_puImage` 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`.

## 14.5 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 graphics library string table functions to automatically decompress them when they are requested from the string table.

The type of compression used varies depending upon the encoding of the strings in the .csv file. By default, it is assumed that the strings are in ASCII, a 7 bit encoding allowing compression by removal of the redundant bit. If the “-u” parameter is passed on the command line, however, mkstringtable will assume 8 bit character encodings and compress only by looking for duplicate substrings. When encoding string tables containing any accented characters or alphabets other than Latin, the “-u” option allows .csv files encoded using UTF-8 or other codepages to be used.

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 [-u] <csvfile> <rootname>
```

Where the arguments mean:

<code>[-u]</code>	Indicates that the .csv file contains UTF8 or some other non-ASCII character encoding. If absent, strings are assumed to be ASCII.
<code>&lt;csvfile&gt;</code>	Specifies the input .csv file to use to create a string table.
<code>&lt;rootname&gt;</code>	Specifies the root name of the output files as <code>&lt;rootname&gt;.c</code> and <code>&lt;rootname&gt;.h</code> . The value is also used in the naming of the string table variable which has a prototype in the <code>&lt;rootname&gt;.h</code> .

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 `grib.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"
Einführung, in Deutsch
Prueba
Verifica
```

Example: String definitions in a .csv file.

```
STR_CONFIG, Configuration, Konfigurieren, Configuración, Configurazione
STR_INTRO, Introduction, Einfhrung, Introducción, Introduzione
STR_QUOTE, Introduction in "English", "Einführung, 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.

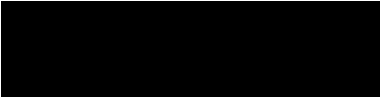






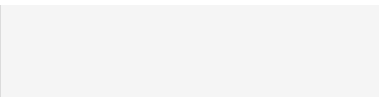











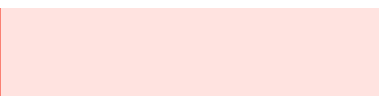







Example: Retrieving a string from the table.

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

## 15 Predefined Color Reference

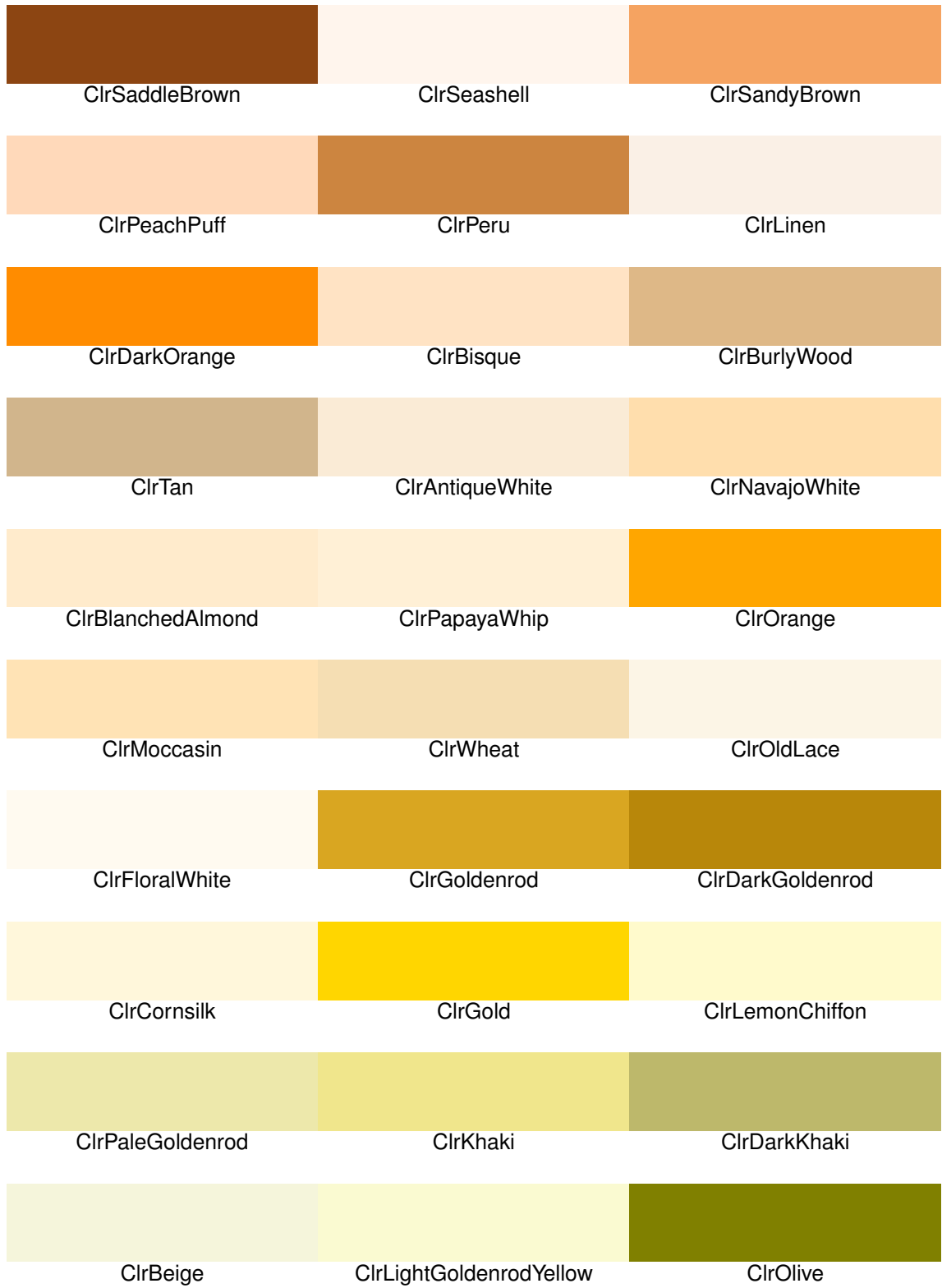
There are a set of predefined colors in `g_rlib.h` that can be used for drawing on the screen. Other colors can be used by specifying their RGB values; the predefined colors are provided as a convenience.

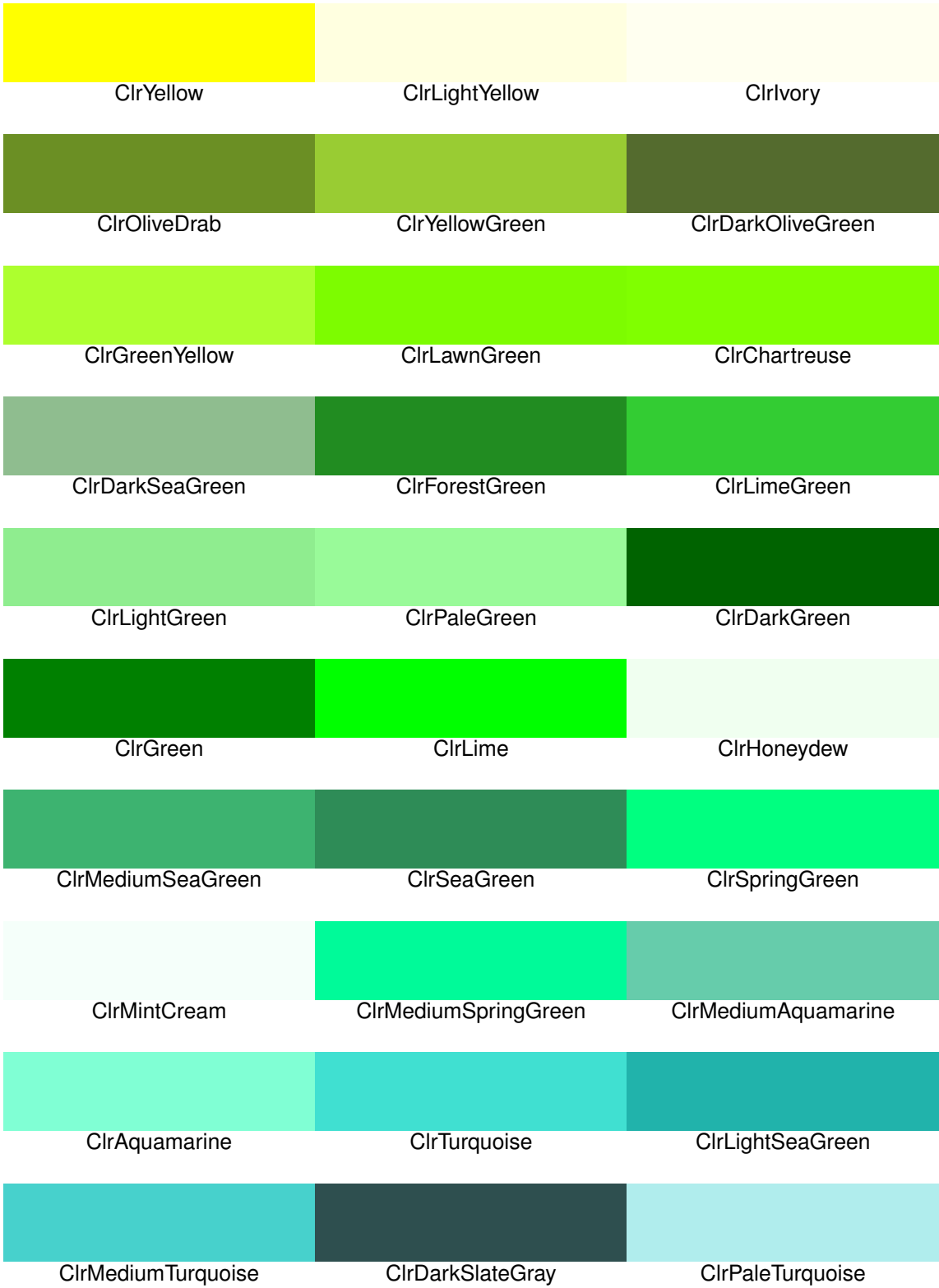
The following table lists the predefined colors, along with a small color swatch showing the color.

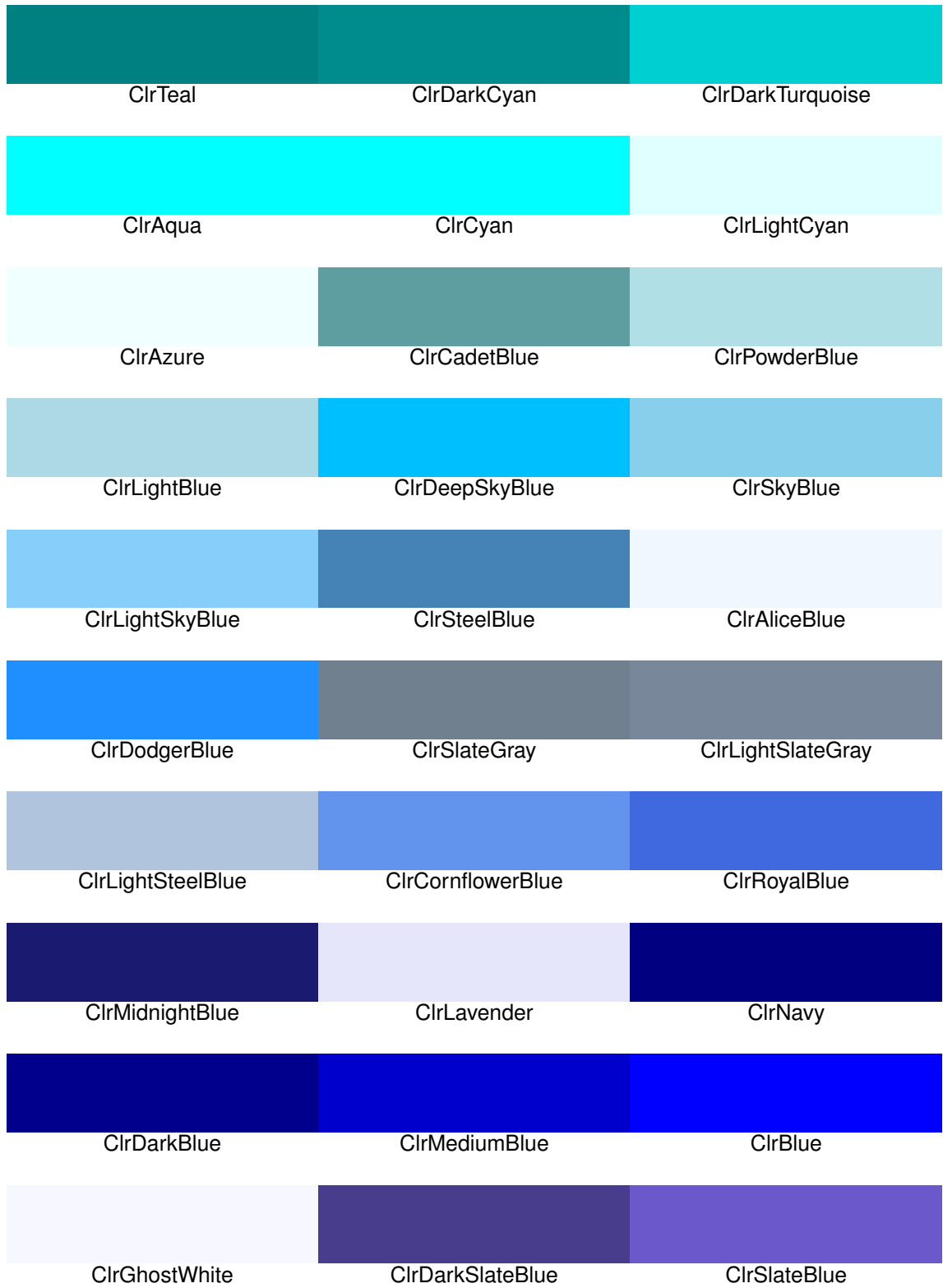
		
ClrBlack	ClrDimGray	ClrGray
		
ClrDarkGray	ClrSilver	ClrLightGrey
		
ClrGainsboro	ClrWhiteSmoke	ClrWhite
		
ClrRosyBrown	ClrIndianRed	ClrBrown
		
ClrFireBrick	ClrLightCoral	ClrMaroon
		
ClrDarkRed	ClrRed	ClrSnow
		
ClrSalmon	ClrMistyRose	ClrTomato
		
ClrDarkSalmon	ClrOrangeRed	ClrCoral
		
ClrLightSalmon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

Predefined Color Reference

---













## 16 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

g\_sFontCm12

g\_sFontCm12b

g\_sFontCm12i



g\_sFontCmsc12

g\_sFontCmss12

g\_sFontCmss12b

g\_sFontCmss12i

g\_sFontCmtt12

g\_sFontCm14

g\_sFontCm14b

g\_sFontCm14i



g\_sFontCmsc14

g\_sFontCmss14

g\_sFontCmss14b

g\_sFontCmss14i

g\_sFontCmtt14

g\_sFontCm16

g\_sFontCm16b

g\_sFontCm16i



g\_sFontCmsc16

g\_sFontCmss16

g\_sFontCmss16b

g\_sFontCmss16i

g\_sFontCmtt16

g\_sFontCm18

g\_sFontCm18b

g\_sFontCm18i



g\_sFontCmsc18

g\_sFontCmss18

g\_sFontCmss18b

g\_sFontCmss18i

g\_sFontCmtt18

g\_sFontCm20

g\_sFontCm20b

g\_sFontCm20i



g\_sFontCmsc20

g\_sFontCmss20

g\_sFontCmss20b

g\_sFontCmss20i

g\_sFontCmtt20

g\_sFontCm22

g\_sFontCm22b

g\_sFontCm22i



g\_sFontCmsc22

g\_sFontCmss22

g\_sFontCmss22b

g\_sFontCmss22i

g\_sFontCmtt22

g\_sFontCm24

g\_sFontCm24b

g\_sFontCm24i



g\_sFontCmsc24

g\_sFontCmss24

g\_sFontCmss24b

g\_sFontCmss24i

g\_sFontCmtt24

g\_sFontCm26

g\_sFontCm26b

g\_sFontCm26i



g\_sFontCmsc26

g\_sFontCmss26

g\_sFontCmss26b

g\_sFontCmss26i

g\_sFontCmtt26

g\_sFontCm28

g\_sFontCm28b

g\_sFontCm28i



g\_sFontCmsc28

g\_sFontCmss28

g\_sFontCmss28b

g\_sFontCmss28i

g\_sFontCmtt28

g\_sFontCm30

g\_sFontCm30b

g\_sFontCm30i



g\_sFontCmsc30

g\_sFontCmss30

g\_sFontCmss30b

g\_sFontCmss30i

g\_sFontCmtt30

g\_sFontCm32

g\_sFontCm32b

g\_sFontCm32i



g\_sFontCmsc32

g\_sFontCmss32

g\_sFontCmss32b

g\_sFontCmss32i

g\_sFontCmtt32

g\_sFontCm34

g\_sFontCm34b

g\_sFontCm34i



g\_sFontCmsc34

g\_sFontCmss34

g\_sFontCmss34b

g\_sFontCmss34i

g\_sFontCmtt34

g\_sFontCm36

g\_sFontCm36b

g\_sFontCm36i



g\_sFontCmsc36

g\_sFontCmss36

g\_sFontCmss36b

g\_sFontCmss36i

g\_sFontCmtt36

g\_sFontCm38

g\_sFontCm38b

g\_sFontCm38i



g\_sFontCmsc38

g\_sFontCmss38

g\_sFontCmss38b

g\_sFontCmss38i

g\_sFontCmtt38

g\_sFontCm40

g\_sFontCm40b

g\_sFontCm40i



g\_sFontCmsc40

g\_sFontCmss40

g\_sFontCmss40b

g\_sFontCmss40i

g\_sFontCmtt40

g\_sFontCm42

g\_sFontCm42b

g\_sFontCm42i



g\_sFontCmsc42

g\_sFontCmss42

g\_sFontCmss42b

g\_sFontCmss42i

g\_sFontCmtt42

g\_sFontCm44

g\_sFontCm44b

g\_sFontCm44i



g\_sFontCmsc44

g\_sFontCmss44

g\_sFontCmss44b

g\_sFontCmss44i

g\_sFontCmtt44

g\_sFontCm46

g\_sFontCm46b

g\_sFontCm46i



g\_sFontCmsc46

g\_sFontCmss46

g\_sFontCmss46b

g\_sFontCmss46i

g\_sFontCmtt46

g\_sFontCm48

g\_sFontCm48b

g\_sFontCm48i



g\_sFontCmsc48

g\_sFontCmss48

g\_sFontCmss48b

g\_sFontCmss48i

g\_sFontCmtt48

---

# IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer’s risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

## Products

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
OMAP Applications Processors	<a href="http://www.ti.com/omap">www.ti.com/omap</a>
Wireless Connectivity	<a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a>

## Applications

Automotive and Transportation	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Communications and Telecom	<a href="http://www.ti.com/communications">www.ti.com/communications</a>
Computers and Peripherals	<a href="http://www.ti.com/computers">www.ti.com/computers</a>
Consumer Electronics	<a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>
Energy and Lighting	<a href="http://www.ti.com/energy">www.ti.com/energy</a>
Industrial	<a href="http://www.ti.com/industrial">www.ti.com/industrial</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Space, Avionics and Defense	<a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a>
Video and Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>

**TI E2E Community** [e2e.ti.com](http://e2e.ti.com)

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2008-2015, Texas Instruments Incorporated