

libg15render

Generated by Doxygen 1.8.8

Fri Oct 17 2014 13:34:49

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	g15canvas Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Field Documentation	5
3.1.2.1	buffer	5
3.1.2.2	ftLib	5
3.1.2.3	mode_cache	6
3.1.2.4	mode_reverse	6
3.1.2.5	mode_xor	6
3.1.2.6	ttf_face	6
3.1.2.7	ttf_fontsize	6
4	File Documentation	7
4.1	config.h File Reference	7
4.1.1	Macro Definition Documentation	7
4.1.1.1	HAVE_DLFCN_H	7
4.1.1.2	HAVE_FT2BUILD_H	7
4.1.1.3	HAVE_INTTYPES_H	8
4.1.1.4	HAVE_LIBG15	8
4.1.1.5	HAVE_LIBM	8
4.1.1.6	HAVE_MEMORY_H	8
4.1.1.7	HAVE_MEMSET	8
4.1.1.8	HAVE_STDINT_H	8
4.1.1.9	HAVE_STDLIB_H	8
4.1.1.10	HAVE_STRING_H	8
4.1.1.11	HAVE_STRINGS_H	8

4.1.1.12	HAVE_SYS_STAT_H	8
4.1.1.13	HAVE_SYS_TYPES_H	8
4.1.1.14	HAVE_UNISTD_H	8
4.1.1.15	PACKAGE	9
4.1.1.16	PACKAGE_BUGREPORT	9
4.1.1.17	PACKAGE_NAME	9
4.1.1.18	PACKAGE_STRING	9
4.1.1.19	PACKAGE_TARNAME	9
4.1.1.20	PACKAGE_VERSION	9
4.1.1.21	STDC_HEADERS	9
4.1.1.22	TTF_SUPPORT	9
4.1.1.23	VERSION	9
4.2	libg15render.h File Reference	9
4.2.1	Macro Definition Documentation	11
4.2.1.1	BYTE_SIZE	11
4.2.1.2	G15_BUFFER_LEN	11
4.2.1.3	G15_COLOR_BLACK	11
4.2.1.4	G15_COLOR_WHITE	12
4.2.1.5	G15_LCD_HEIGHT	12
4.2.1.6	G15_LCD_OFFSET	12
4.2.1.7	G15_LCD_WIDTH	12
4.2.1.8	G15_MAX_FACE	12
4.2.1.9	G15_PIXEL_FILL	12
4.2.1.10	G15_PIXEL_NOFILL	12
4.2.1.11	G15_TEXT_LARGE	12
4.2.1.12	G15_TEXT_MED	12
4.2.1.13	G15_TEXT_SMALL	12
4.2.2	Typedef Documentation	13
4.2.2.1	g15canvas	13
4.2.3	Function Documentation	13
4.2.3.1	g15r_clearScreen	13
4.2.3.2	g15r_drawBar	13
4.2.3.3	g15r_drawBigNum	14
4.2.3.4	g15r_drawCircle	15
4.2.3.5	g15r_drawIcon	16
4.2.3.6	g15r_drawLine	16
4.2.3.7	g15r_drawRoundBox	17
4.2.3.8	g15r_drawSprite	18
4.2.3.9	g15r_getPixel	19
4.2.3.10	g15r_initCanvas	19

4.2.3.11	g15r_loadWbmpSplash	20
4.2.3.12	g15r_loadWbmpToBuf	20
4.2.3.13	g15r_pixelBox	21
4.2.3.14	g15r_pixelOverlay	22
4.2.3.15	g15r_pixelReverseFill	22
4.2.3.16	g15r_renderCharacterLarge	24
4.2.3.17	g15r_renderCharacterMedium	24
4.2.3.18	g15r_renderCharacterSmall	25
4.2.3.19	g15r_renderString	25
4.2.3.20	g15r_setPixel	26
4.2.3.21	g15r_ttfLoad	26
4.2.3.22	g15r_ttfPrint	28
4.2.4	Variable Documentation	29
4.2.4.1	fontdata_6x4	29
4.2.4.2	fontdata_7x5	29
4.2.4.3	fontdata_8x8	29
4.3	pixel.c File Reference	29
4.3.1	Function Documentation	30
4.3.1.1	g15r_drawBar	30
4.3.1.2	g15r_drawBigNum	31
4.3.1.3	g15r_drawCircle	32
4.3.1.4	g15r_drawIcon	33
4.3.1.5	g15r_drawLine	33
4.3.1.6	g15r_drawRoundBox	34
4.3.1.7	g15r_drawSprite	35
4.3.1.8	g15r_loadWbmpSplash	36
4.3.1.9	g15r_loadWbmpToBuf	36
4.3.1.10	g15r_pixelBox	37
4.3.1.11	g15r_pixelOverlay	38
4.3.1.12	g15r_pixelReverseFill	39
4.3.1.13	swap	39
4.4	screen.c File Reference	39
4.4.1	Function Documentation	40
4.4.1.1	g15r_clearScreen	40
4.4.1.2	g15r_getPixel	40
4.4.1.3	g15r_initCanvas	40
4.4.1.4	g15r_setPixel	41
4.5	text.c File Reference	41
4.5.1	Function Documentation	42
4.5.1.1	calc_ttf_centering	42

4.5.1.2	calc_ttf_right_justify	42
4.5.1.3	calc_ttf_totalstringwidth	43
4.5.1.4	calc_ttf_true_ypos	43
4.5.1.5	draw_ttf_char	43
4.5.1.6	draw_ttf_str	44
4.5.1.7	g15r_renderCharacterLarge	44
4.5.1.8	g15r_renderCharacterMedium	44
4.5.1.9	g15r_renderCharacterSmall	45
4.5.1.10	g15r_renderString	45
4.5.1.11	g15r_ttfLoad	46
4.5.1.12	g15r_ttfPrint	46

Index	49
--------------	-----------

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

g15canvas	
This structure holds the data need to render objects to the LCD screen	5

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

config.h	7
libg15render.h	9
pixel.c	29
screen.c	39
text.c	41

Chapter 3

Data Structure Documentation

3.1 g15canvas Struct Reference

This structure holds the data need to render objects to the LCD screen.

```
#include <libg15render.h>
```

Data Fields

- unsigned char **buffer** [G15_BUFFER_LEN]
- FT_Library **ftLib**
- int **mode_cache**
- int **mode_reverse**
- int **mode_xor**
- FT_Face **ttf_face** [G15_MAX_FACE][sizeof(FT_Face)]
- int **ttf_fontsize** [G15_MAX_FACE]

3.1.1 Detailed Description

This structure holds the data need to render objects to the LCD screen.

Definition at line 36 of file libg15render.h.

3.1.2 Field Documentation

3.1.2.1 unsigned char g15canvas::buffer[G15_BUFFER_LEN]

g15canvas::buffer (p. 5)[] is a buffer holding the pixel data to be sent to the LCD.

Definition at line 39 of file libg15render.h.

Referenced by g15r_clearScreen(), g15r_getPixel(), g15r_initCanvas(), g15r_loadWbmpSplash(), and g15r_setPixel().

3.1.2.2 FT_Library g15canvas::ftLib

Definition at line 47 of file libg15render.h.

Referenced by draw_ttf_char(), g15r_initCanvas(), and g15r_ttfLoad().

3.1.2.3 `int g15canvas::mode_cache`

`g15canvas::mode_cache` (p. 6) can be used to determine whether caching should be used in an application.

Definition at line 43 of file `libg15render.h`.

Referenced by `g15r_initCanvas()`.

3.1.2.4 `int g15canvas::mode_reverse`

`g15canvas::mode_reverse` (p. 6) determines whether color values passed to `g15r_setPixel` are reversed.

Definition at line 45 of file `libg15render.h`.

Referenced by `g15r_initCanvas()`, and `g15r_setPixel()`.

3.1.2.5 `int g15canvas::mode_xor`

`g15canvas::mode_xor` (p. 6) determines whether xor processing is used in `g15r_setPixel`.

Definition at line 41 of file `libg15render.h`.

Referenced by `g15r_initCanvas()`, and `g15r_setPixel()`.

3.1.2.6 `FT_Face g15canvas::ttf_face[G15_MAX_FACE][sizeof(FT_Face)]`

Definition at line 48 of file `libg15render.h`.

Referenced by `g15r_ttfLoad()`, and `g15r_ttfPrint()`.

3.1.2.7 `int g15canvas::ttf_fontsize[G15_MAX_FACE]`

Definition at line 49 of file `libg15render.h`.

Referenced by `g15r_ttfLoad()`, and `g15r_ttfPrint()`.

The documentation for this struct was generated from the following file:

- **`libg15render.h`**

Chapter 4

File Documentation

4.1 config.h File Reference

Macros

- `#define HAVE_DLFCN_H 1`
- `#define HAVE_FT2BUILD_H 1`
- `#define HAVE_INTTYPES_H 1`
- `#define HAVE_LIBG15 1`
- `#define HAVE_LIBM 1`
- `#define HAVE_MEMORY_H 1`
- `#define HAVE_MEMSET 1`
- `#define HAVE_STDINT_H 1`
- `#define HAVE_STDLIB_H 1`
- `#define HAVE_STRING_H 1`
- `#define HAVE_STRINGS_H 1`
- `#define HAVE_SYS_STAT_H 1`
- `#define HAVE_SYS_TYPES_H 1`
- `#define HAVE_UNISTD_H 1`
- `#define PACKAGE "libg15render"`
- `#define PACKAGE_BUGREPORT "mirabeaj@gmail.com"`
- `#define PACKAGE_NAME "libg15render"`
- `#define PACKAGE_STRING "libg15render 1.2"`
- `#define PACKAGE_TARNAME "libg15render"`
- `#define PACKAGE_VERSION "1.2"`
- `#define STDC_HEADERS 1`
- `#define TTF_SUPPORT 1`
- `#define VERSION "1.2"`

4.1.1 Macro Definition Documentation

4.1.1.1 `#define HAVE_DLFCN_H 1`

Definition at line 5 of file config.h.

4.1.1.2 `#define HAVE_FT2BUILD_H 1`

Definition at line 8 of file config.h.

4.1.1.3 `#define HAVE_INTTYPES_H 1`

Definition at line 11 of file config.h.

4.1.1.4 `#define HAVE_LIBG15 1`

Definition at line 14 of file config.h.

4.1.1.5 `#define HAVE_LIBM 1`

Definition at line 17 of file config.h.

4.1.1.6 `#define HAVE_MEMORY_H 1`

Definition at line 20 of file config.h.

4.1.1.7 `#define HAVE_MEMSET 1`

Definition at line 23 of file config.h.

4.1.1.8 `#define HAVE_STDINT_H 1`

Definition at line 26 of file config.h.

4.1.1.9 `#define HAVE_STDLIB_H 1`

Definition at line 29 of file config.h.

4.1.1.10 `#define HAVE_STRING_H 1`

Definition at line 35 of file config.h.

4.1.1.11 `#define HAVE_STRINGS_H 1`

Definition at line 32 of file config.h.

4.1.1.12 `#define HAVE_SYS_STAT_H 1`

Definition at line 38 of file config.h.

4.1.1.13 `#define HAVE_SYS_TYPES_H 1`

Definition at line 41 of file config.h.

4.1.1.14 `#define HAVE_UNISTD_H 1`

Definition at line 44 of file config.h.

4.1.1.15 `#define PACKAGE "libg15render"`

Definition at line 47 of file config.h.

4.1.1.16 `#define PACKAGE_BUGREPORT "mirabeaj@gmail.com"`

Definition at line 50 of file config.h.

4.1.1.17 `#define PACKAGE_NAME "libg15render"`

Definition at line 53 of file config.h.

4.1.1.18 `#define PACKAGE_STRING "libg15render 1.2"`

Definition at line 56 of file config.h.

4.1.1.19 `#define PACKAGE_TARNAME "libg15render"`

Definition at line 59 of file config.h.

4.1.1.20 `#define PACKAGE_VERSION "1.2"`

Definition at line 62 of file config.h.

4.1.1.21 `#define STDC_HEADERS 1`

Definition at line 65 of file config.h.

4.1.1.22 `#define TTF_SUPPORT 1`

Definition at line 68 of file config.h.

4.1.1.23 `#define VERSION "1.2"`

Definition at line 71 of file config.h.

4.2 libg15render.h File Reference

```
#include <string.h>
#include <ft2build.h>
```

Data Structures

- struct **g15canvas**

This structure holds the data need to render objects to the LCD screen.

Macros

- #define **BYTE_SIZE** 8
- #define **G15_BUFFER_LEN** 1048
- #define **G15_COLOR_BLACK** 1
- #define **G15_COLOR_WHITE** 0
- #define **G15_LCD_HEIGHT** 43
- #define **G15_LCD_OFFSET** 32
- #define **G15_LCD_WIDTH** 160
- #define **G15_MAX_FACE** 5
- #define **G15_PIXEL_FILL** 1
- #define **G15_PIXEL_NOFILL** 0
- #define **G15_TEXT_LARGE** 2
- #define **G15_TEXT_MED** 1
- #define **G15_TEXT_SMALL** 0

Typedefs

- typedef struct **g15canvas** **g15canvas**
This structure holds the data need to render objects to the LCD screen.

Functions

- void **g15r_clearScreen** (**g15canvas** *canvas, int color)
Fills the screen with pixels of color.
- void **g15r_drawBar** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int color, int num, int max, int type)
Draws a completion bar.
- void **g15r_drawBigNum** (**g15canvas** *canvas, unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int color, int num)
Draw a large number.
- void **g15r_drawCircle** (**g15canvas** *canvas, int x, int y, int r, int fill, int color)
Draws a circle centered at (x, y) with a radius of r.
- void **g15r_drawIcon** (**g15canvas** *canvas, char *buf, int my_x, int my_y, int width, int height)
Draw an icon to the screen from a wbmp buffer.
- void **g15r_drawLine** (**g15canvas** *canvas, int px1, int py1, int px2, int py2, const int color)
Draws a line from (px1, py1) to (px2, py2)
- void **g15r_drawRoundBox** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int fill, int color)
Draws a box with rounded corners bounded by (x1, y1) and (x2, y2)
- void **g15r_drawSprite** (**g15canvas** *canvas, char *buf, int my_x, int my_y, int width, int height, int start_x, int start_y, int total_width)
Draw a sprite to the screen from a wbmp buffer.
- int **g15r_getPixel** (**g15canvas** *canvas, unsigned int x, unsigned int y)
Gets the value of the pixel at (x, y)
- void **g15r_initCanvas** (**g15canvas** *canvas)
Clears the canvas and resets the mode switches.
- int **g15r_loadWbmpSplash** (**g15canvas** *canvas, char *filename)
Draw a splash screen from 160x43 wbmp file.
- char * **g15r_loadWbmpToBuf** (char *filename, int *img_width, int *img_height)
Load a wbmp file into a buffer.
- void **g15r_pixelBox** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int color, int thick, int fill)
Draws a box bounded by (x1, y1) and (x2, y2)
- void **g15r_pixelOverlay** (**g15canvas** *canvas, int x1, int y1, int width, int height, short colormap[])

Overlays a bitmap of size width x height starting at (x1, y1)

- void **g15r_pixelReverseFill** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int fill, int color)

Fills an area bounded by (x1, y1) and (x2, y2)

- void **g15r_renderCharacterLarge** (**g15canvas** *canvas, int x, int y, unsigned char character, unsigned int sx, unsigned int sy)

Renders a character in the large font at (x, y)

- void **g15r_renderCharacterMedium** (**g15canvas** *canvas, int x, int y, unsigned char character, unsigned int sx, unsigned int sy)

Renders a character in the meduim font at (x, y)

- void **g15r_renderCharacterSmall** (**g15canvas** *canvas, int x, int y, unsigned char character, unsigned int sx, unsigned int sy)

Renders a character in the small font at (x, y)

- void **g15r_renderString** (**g15canvas** *canvas, unsigned char stringOut[], int row, int size, unsigned int sx, unsigned int sy)

Renders a string with font size in row.

- void **g15r_setPixel** (**g15canvas** *canvas, unsigned int x, unsigned int y, int val)

Sets the value of the pixel at (x, y)

- void **g15r_ttfLoad** (**g15canvas** *canvas, char *fontname, int fontsize, int face_num)

Loads a font through the FreeType2 library.

- void **g15r_ttfPrint** (**g15canvas** *canvas, int x, int y, int fontsize, int face_num, int color, int center, char *print_string)

Prints a string in a given font.

Variables

- unsigned char **fontdata_6x4** []

Font data for the small (6x4) font.

- unsigned char **fontdata_7x5** []

Font data for the medium (7x5) font.

- unsigned char **fontdata_8x8** []

Font data for the large (8x8) font.

4.2.1 Macro Definition Documentation

4.2.1.1 #define BYTE_SIZE 8

Definition at line 21 of file libg15render.h.

Referenced by g15r_drawlcon(), g15r_drawSprite(), g15r_getPixel(), g15r_loadWbmpToBuf(), and g15r_setPixel().

4.2.1.2 #define G15_BUFFER_LEN 1048

Definition at line 22 of file libg15render.h.

Referenced by g15r_clearScreen(), g15r_initCanvas(), and g15r_loadWbmpSplash().

4.2.1.3 #define G15_COLOR_BLACK 1

Definition at line 27 of file libg15render.h.

Referenced by g15r_drawRoundBox(), g15r_pixelOverlay(), g15r_renderCharacterLarge(), g15r_renderCharacterMedium(), and g15r_renderCharacterSmall().

4.2.1.4 `#define G15_COLOR_WHITE 0`

Definition at line 26 of file libg15render.h.

Referenced by `g15r_drawRoundBox()`, `g15r_pixelOverlay()`, `g15r_renderCharacterLarge()`, `g15r_renderCharacterMedium()`, and `g15r_renderCharacterSmall()`.

4.2.1.5 `#define G15_LCD_HEIGHT 43`

Definition at line 24 of file libg15render.h.

Referenced by `g15r_getPixel()`, and `g15r_setPixel()`.

4.2.1.6 `#define G15_LCD_OFFSET 32`

Definition at line 23 of file libg15render.h.

4.2.1.7 `#define G15_LCD_WIDTH 160`

Definition at line 25 of file libg15render.h.

Referenced by `g15r_getPixel()`, and `g15r_setPixel()`.

4.2.1.8 `#define G15_MAX_FACE 5`

Definition at line 33 of file libg15render.h.

Referenced by `g15r_ttfLoad()`.

4.2.1.9 `#define G15_PIXEL_FILL 1`

Definition at line 32 of file libg15render.h.

4.2.1.10 `#define G15_PIXEL_NOFILL 0`

Definition at line 31 of file libg15render.h.

4.2.1.11 `#define G15_TEXT_LARGE 2`

Definition at line 30 of file libg15render.h.

Referenced by `g15r_renderString()`.

4.2.1.12 `#define G15_TEXT_MED 1`

Definition at line 29 of file libg15render.h.

Referenced by `g15r_renderString()`.

4.2.1.13 `#define G15_TEXT_SMALL 0`

Definition at line 28 of file libg15render.h.

Referenced by `g15r_renderString()`.

4.2.2 Typedef Documentation

4.2.2.1 typedef struct g15canvas g15canvas

This structure holds the data need to render objects to the LCD screen.

4.2.3 Function Documentation

4.2.3.1 void g15r_clearScreen (g15canvas * canvas, int color)

Fills the screen with pixels of color.

Clears the screen and fills it with pixels of color

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>color</i>	Screen will be filled with this color.

Definition at line 80 of file screen.c.

References g15canvas::buffer, and G15_BUFFER_LEN.

```
81 {
82     memset (canvas->buffer, (color ? 0xFF : 0), G15_BUFFER_LEN);
83 }
```

4.2.3.2 void g15r_drawBar (g15canvas * canvas, int x1, int y1, int x2, int y2, int color, int num, int max, int type)

Draws a completion bar.

Given a maximum value, and a value between 0 and that maximum value, calculate and draw a bar showing that percentage.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of the bar.
<i>y1</i>	Defines uppermost bound of the bar.
<i>x2</i>	Defines rightmost bound of the bar.
<i>y2</i>	Defines bottommost bound of the bar.
<i>color</i>	The bar will be drawn this color.
<i>num</i>	Number of units relative to max filled.
<i>max</i>	Number of units equal to 100% filled.
<i>type</i>	Type of bar. 1=solid bar, 2=solid bar with border, 3 = solid bar with l-frame.

Definition at line 337 of file pixel.c.

References g15r_drawLine(), and g15r_pixelBox().

```
339 {
340     float len, length;
341     int x;
342     if (max == 0)
343         return;
344     if (num > max)
345         num = max;
346
347     if (type == 2)
348     {
349         y1 += 2;
350         y2 -= 2;
351         x1 += 2;
352         x2 -= 2;
353     }
354 }
```

```

355 len = ((float) max / (float) num);
356 length = (x2 - x1) / len;
357
358 if (type == 1)
359 {
360     gl5r_pixelBox (canvas, x1, y1 - type, x2, y2 + type, color ^ 1, 1, 1);
361     gl5r_pixelBox (canvas, x1, y1 - type, x2, y2 + type, color, 1, 0);
362 }
363 else if (type == 2)
364 {
365     gl5r_pixelBox (canvas, x1 - 2, y1 - type, x2 + 2, y2 + type, color ^ 1,
366                   1, 1);
367     gl5r_pixelBox (canvas, x1 - 2, y1 - type, x2 + 2, y2 + type, color, 1,
368                   0);
369 }
370 else if (type == 3)
371 {
372     gl5r_drawLine (canvas, x1, y1 - type, x1, y2 + type, color);
373     gl5r_drawLine (canvas, x2, y1 - type, x2, y2 + type, color);
374     gl5r_drawLine (canvas, x1, y1 + ((y2 - y1) / 2), x2,
375                   y1 + ((y2 - y1) / 2), color);
376 }
377 gl5r_pixelBox (canvas, x1, y1, (int) ceil (x1 + length), y2, color, 1, 1);
378 }

```

4.2.3.3 void gl5r_drawBigNum (g15canvas * canvas, unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int color, int num)

Draw a large number.

Draw a large number to a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p.5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of the number.
<i>y1</i>	Defines uppermost bound of the number.
<i>x2</i>	Defines rightmost bound of the number.
<i>y2</i>	Defines bottommost bound of the number.
<i>num</i>	The number to be drawn.

Definition at line 545 of file pixel.c.

References gl5r_pixelBox().

```

546 {
547     x1 += 2;
548     x2 -= 2;
549
550     switch(num) {
551         case 0:
552             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
553             gl5r_pixelBox (canvas, x1 +5, y1 +5, x2 -5, y2 - 6, 1 - color, 1, 1);
554             break;
555         case 1:
556             gl5r_pixelBox (canvas, x2-5, y1, x2, y2 , color, 1, 1);
557             gl5r_pixelBox (canvas, x1, y1, x2 -5, y2, 1 - color, 1, 1);
558             break;
559         case 2:
560             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
561             gl5r_pixelBox (canvas, x1, y1+5, x2 -5, y1+((y2/2)-3), 1 - color, 1, 1);
562             gl5r_pixelBox (canvas, x1+5, y1+((y2/2)+3), x2 , y2-6, 1 - color, 1, 1);
563             break;
564         case 3:
565             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
566             gl5r_pixelBox (canvas, x1, y1+5, x2 -5, y1+((y2/2)-3), 1 - color, 1, 1);
567             gl5r_pixelBox (canvas, x1, y1+((y2/2)+3), x2-5 , y2-6, 1 - color, 1, 1);
568             break;
569         case 4:
570             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
571             gl5r_pixelBox (canvas, x1, y1+((y2/2)+3), x2 -5, y2, 1 - color, 1, 1);
572             gl5r_pixelBox (canvas, x1+5, y1, x2-5 , y1+((y2/2)-3), 1 - color, 1, 1);
573             break;
574         case 5:
575             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
576             gl5r_pixelBox (canvas, x1+5, y1+5, x2 , y1+((y2/2)-3), 1 - color, 1, 1);
577             gl5r_pixelBox (canvas, x1, y1+((y2/2)+3), x2-5 , y2-6, 1 - color, 1, 1);

```

```

578         break;
579     case 6:
580         gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
581         gl5r_pixelBox (canvas, x1+5, y1+5, x2 , y1+((y2/2)-3), 1 - color, 1, 1);
582         gl5r_pixelBox (canvas, x1+5, y1+((y2/2)+3), x2-5 , y2-6, 1 - color, 1, 1);
583         break;
584     case 7:
585         gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
586         gl5r_pixelBox (canvas, x1, y1+5, x2 -5, y2, 1 - color, 1, 1);
587         break;
588     case 8:
589         gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
590         gl5r_pixelBox (canvas, x1+5, y1+5, x2-5 , y1+((y2/2)-3), 1 - color, 1, 1);
591         gl5r_pixelBox (canvas, x1+5, y1+((y2/2)+3), x2-5 , y2-6, 1 - color, 1, 1);
592         break;
593     case 9:
594         gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
595         gl5r_pixelBox (canvas, x1+5, y1+5, x2-5 , y1+((y2/2)-3), 1 - color, 1, 1);
596         gl5r_pixelBox (canvas, x1, y1+((y2/2)+3), x2-5 , y2, 1 - color, 1, 1);
597         break;
598     case 10:
599         gl5r_pixelBox (canvas, x2-5, y1+5, x2, y1+10 , color, 1, 1);
600         gl5r_pixelBox (canvas, x2-5, y2-10, x2, y2-5 , color, 1, 1);
601         break;
602     case 11:
603         gl5r_pixelBox (canvas, x1, y1+((y2/2)-2), x2, y1+((y2/2)+2), color, 1, 1);
604         break;
605     case 12:
606         gl5r_pixelBox (canvas, x2-5, y2-5, x2, y2 , color, 1, 1);
607         break;
608     }
609 }

```

4.2.3.4 void gl5r_drawCircle (g15canvas * canvas, int x, int y, int r, int fill, int color)

Draws a circle centered at (x, y) with a radius of r.

Draws a circle centered at (x, y) with a radius of r.

The circle will be filled if fill != 0.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x</i>	Defines horizontal center of the circle.
<i>y</i>	Defines vertical center of circle.
<i>r</i>	Defines radius of circle.
<i>fill</i>	The circle will be filled with color if fill != 0.
<i>color</i>	Lines defining the circle will be drawn this color.

Definition at line 203 of file pixel.c.

References `gl5r_drawLine()`, and `gl5r_setPixel()`.

```

204 {
205     int xx, yy, dd;
206
207     xx = 0;
208     yy = r;
209     dd = 2 * (1 - r);
210
211     while (yy >= 0)
212     {
213         if (!fill)
214         {
215             gl5r_setPixel (canvas, x + xx, y - yy, color);
216             gl5r_setPixel (canvas, x + xx, y + yy, color);
217             gl5r_setPixel (canvas, x - xx, y - yy, color);
218             gl5r_setPixel (canvas, x - xx, y + yy, color);
219         }
220         else
221         {
222             gl5r_drawLine (canvas, x - xx, y - yy, x + xx, y - yy, color);
223             gl5r_drawLine (canvas, x - xx, y + yy, x + xx, y + yy, color);
224         }
225         if (dd + yy > 0)
226         {
227             yy--;

```

```

228         dd = dd - (2 * yy + 1);
229     }
230     if (xx > dd)
231     {
232         xx++;
233         dd = dd + (2 * xx + 1);
234     }
235 }
236 }

```

4.2.3.5 void g15r_drawIcon (g15canvas * canvas, char * buf, int my_x, int my_y, int width, int height)

Draw an icon to the screen from a wbmp buffer.

Draw an icon to a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated in is found.
<i>buf</i>	A pointer to the buffer holding the icon to be displayed.
<i>my_x</i>	Leftmost boundary of image.
<i>my_y</i>	Topmost boundary of image.
<i>width</i>	Width of the image in buf.
<i>height</i>	Height of the image in buf.

Definition at line 411 of file pixel.c.

References `BYTE_SIZE`, and `g15r_setPixel()`.

```

412 {
413     int y,x,val;
414     unsigned int pixel_offset = 0;
415     unsigned int byte_offset, bit_offset;
416
417     for (y=0; y < height - 1; y++)
418         for (x=0; x < width - 1; x++)
419         {
420             pixel_offset = y * width + x;
421             byte_offset = pixel_offset / BYTE_SIZE;
422             bit_offset = 7 - (pixel_offset % BYTE_SIZE);
423
424             val = (buf[byte_offset] & (1 << bit_offset)) >> bit_offset;
425             g15r_setPixel (canvas, x + my_x, y + my_y, val);
426         }
427 }

```

4.2.3.6 void g15r_drawLine (g15canvas * canvas, int px1, int py1, int px2, int py2, const int color)

Draws a line from (px1, py1) to (px2, py2)

A line of color is drawn from (px1, py1) to (px2, py2).

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>px1</i>	X component of point 1.
<i>py1</i>	Y component of point 1.
<i>px2</i>	X component of point 2.
<i>py2</i>	Y component of point 2.
<i>color</i>	Line will be drawn this color.

Definition at line 99 of file pixel.c.

References `g15r_setPixel()`, and `swap()`.

Referenced by `g15r_drawBar()`, `g15r_drawCircle()`, `g15r_drawRoundBox()`, and `g15r_pixelBox()`.

```

101 {

```

```

102  /*
103   * Bresenham's Line Algorithm
104   * http://en.wikipedia.org/wiki/Bresenham's\_algorithm
105   */
106
107  int steep = 0;
108
109  if (abs (py2 - py1) > abs (px2 - px1))
110      steep = 1;
111
112  if (steep)
113  {
114      swap (&px1, &py1);
115      swap (&px2, &py2);
116  }
117
118  if (px1 > px2)
119  {
120      swap (&px1, &px2);
121      swap (&py1, &py2);
122  }
123
124  int dx = px2 - px1;
125  int dy = abs (py2 - py1);
126
127  int error = 0;
128  int y = py1;
129  int ystep = (py1 < py2) ? 1 : -1;
130  int x = 0;
131
132  for (x = px1; x <= px2; ++x)
133  {
134      if (steep)
135          g15r_setPixel (canvas, y, x, color);
136      else
137          g15r_setPixel (canvas, x, y, color);
138
139      error += dy;
140      if (2 * error >= dx)
141      {
142          y += ystep;
143          error -= dx;
144      }
145  }
146 }

```

4.2.3.7 void g15r_drawRoundBox (g15canvas * canvas, int x1, int y1, int x2, int y2, int fill, int color)

Draws a box with rounded corners bounded by (x1, y1) and (x2, y2)

Draws a rounded box around the area bounded by (x1, y1) and (x2, y2).

The box will be filled if fill != 0.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of the box.
<i>y1</i>	Defines uppermost bound of the box.
<i>x2</i>	Defines rightmost bound of the box.
<i>y2</i>	Defines bottommost bound of the box.
<i>fill</i>	The box will be filled with color if fill != 0.
<i>color</i>	Lines defining the box will be drawn this color.

Definition at line 252 of file pixel.c.

References G15_COLOR_BLACK, G15_COLOR_WHITE, g15r_drawLine(), and g15r_setPixel().

```

254 {
255     int y, shave = 3;
256
257     if (shave > (x2 - x1) / 2)
258         shave = (x2 - x1) / 2;
259     if (shave > (y2 - y1) / 2)
260         shave = (y2 - y1) / 2;
261
262     if ((x1 != x2) && (y1 != y2))

```

```

263 {
264     if (fill)
265     {
266         gl5r_drawLine (canvas, x1 + shave, y1, x2 - shave, y1, color);
267         for (y = y1 + 1; y < y1 + shave; y++)
268             gl5r_drawLine (canvas, x1 + 1, y, x2 - 1, y, color);
269         for (y = y1 + shave; y <= y2 - shave; y++)
270             gl5r_drawLine (canvas, x1, y, x2, y, color);
271         for (y = y2 - shave + 1; y < y2; y++)
272             gl5r_drawLine (canvas, x1 + 1, y, x2 - 1, y, color);
273         gl5r_drawLine (canvas, x1 + shave, y2, x2 - shave, y2, color);
274         if (shave == 4)
275         {
276             gl5r_setPixel (canvas, x1 + 1, y1 + 1,
277                             color ==
278                             G15_COLOR_WHITE ? G15_COLOR_BLACK :
279                             G15_COLOR_WHITE);
280             gl5r_setPixel (canvas, x1 + 1, y2 - 1,
281                             color ==
282                             G15_COLOR_WHITE ? G15_COLOR_BLACK :
283                             G15_COLOR_WHITE);
284             gl5r_setPixel (canvas, x2 - 1, y1 + 1,
285                             color ==
286                             G15_COLOR_WHITE ? G15_COLOR_BLACK :
287                             G15_COLOR_WHITE);
288             gl5r_setPixel (canvas, x2 - 1, y2 - 1,
289                             color ==
290                             G15_COLOR_WHITE ? G15_COLOR_BLACK :
291                             G15_COLOR_WHITE);
292         }
293     }
294     else
295     {
296         gl5r_drawLine (canvas, x1 + shave, y1, x2 - shave, y1, color);
297         gl5r_drawLine (canvas, x1, y1 + shave, x1, y2 - shave, color);
298         gl5r_drawLine (canvas, x2, y1 + shave, x2, y2 - shave, color);
299         gl5r_drawLine (canvas, x1 + shave, y2, x2 - shave, y2, color);
300         if (shave > 1)
301         {
302             gl5r_drawLine (canvas, x1 + 1, y1 + 1, x1 + shave - 1, y1 + 1,
303                             color);
304             gl5r_drawLine (canvas, x2 - shave + 1, y1 + 1, x2 - 1, y1 + 1,
305                             color);
306             gl5r_drawLine (canvas, x1 + 1, y2 - 1, x1 + shave - 1, y2 - 1,
307                             color);
308             gl5r_drawLine (canvas, x2 - shave + 1, y2 - 1, x2 - 1, y2 - 1,
309                             color);
310             gl5r_drawLine (canvas, x1 + 1, y1 + 1, x1 + 1, y1 + shave - 1,
311                             color);
312             gl5r_drawLine (canvas, x1 + 1, y2 - 1, x1 + 1, y2 - shave + 1,
313                             color);
314             gl5r_drawLine (canvas, x2 - 1, y1 + 1, x2 - 1, y1 + shave - 1,
315                             color);
316             gl5r_drawLine (canvas, x2 - 1, y2 - 1, x2 - 1, y2 - shave + 1,
317                             color);
318         }
319     }
320 }
321 }

```

4.2.3.8 void `gl5r_drawSprite (g15canvas * canvas, char * buf, int my_x, int my_y, int width, int height, int start_x, int start_y, int total_width)`

Draw a sprite to the screen from a wbmp buffer.

Draw a sprite to a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated in is found.
<i>buf</i>	A pointer to the buffer holding a set of sprites.
<i>my_x</i>	Leftmost boundary of image.
<i>my_y</i>	Topmost boundary of image.

<i>width</i>	Width of the sprite.
<i>height</i>	Height of the sprite.
<i>start_x</i>	X offset for reading sprite from buf.
<i>start_y</i>	Y offset for reading sprite from buf.
<i>total_width</i>	Width of the set of sprites held in buf.

Definition at line 443 of file pixel.c.

References `BYTE_SIZE`, and `g15r_setPixel()`.

```

444 {
445     int y,x,val;
446     unsigned int pixel_offset = 0;
447     unsigned int byte_offset, bit_offset;
448
449     for (y=0; y < height - 1; y++)
450         for (x=0; x < width - 1; x++)
451         {
452             pixel_offset = (y + start_y) * total_width + (x + start_x);
453             byte_offset = pixel_offset / BYTE_SIZE;
454             bit_offset = 7 - (pixel_offset % BYTE_SIZE);
455
456             val = (buf[byte_offset] & (1 << bit_offset)) >> bit_offset;
457             g15r_setPixel (canvas, x + my_x, y + my_y, val);
458         }
459 }
```

4.2.3.9 int g15r_getPixel (g15canvas * canvas, unsigned int x, unsigned int y)

Gets the value of the pixel at (x, y)

Retrieves the value of the pixel at (x, y)

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x</i>	X offset for pixel to be retrieved.
<i>y</i>	Y offset for pixel to be retrieved.

Definition at line 29 of file screen.c.

References `g15canvas::buffer`, `BYTE_SIZE`, `G15_LCD_HEIGHT`, and `G15_LCD_WIDTH`.

Referenced by `g15r_pixelReverseFill()`, and `g15r_setPixel()`.

```

30 {
31     if (x >= G15_LCD_WIDTH || y >= G15_LCD_HEIGHT)
32         return 0;
33
34     unsigned int pixel_offset = y * G15_LCD_WIDTH + x;
35     unsigned int byte_offset = pixel_offset / BYTE_SIZE;
36     unsigned int bit_offset = 7 - (pixel_offset % BYTE_SIZE);
37
38     return (canvas->buffer[byte_offset] & (1 << bit_offset)) >> bit_offset;
39 }
```

4.2.3.10 void g15r_initCanvas (g15canvas * canvas)

Clears the canvas and resets the mode switches.

Clears the screen and resets the mode values for a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct
---------------	---

Definition at line 91 of file screen.c.

References `g15canvas::buffer`, `g15canvas::ftLib`, `G15_BUFFER_LEN`, `g15canvas::mode_cache`, `g15canvas::mode_reverse`, and `g15canvas::mode_xor`.

```

92 {
93     memset (canvas->buffer, 0, G15_BUFFER_LEN);
94     canvas->mode_cache = 0;
95     canvas->mode_reverse = 0;
96     canvas->mode_xor = 0;
97 #ifdef TTF_SUPPORT
98     if (FT_Init_FreeType (&canvas->ftLib))
99         printf ("Freetype couldnt initialise\n");
100 #endif
101 }
```

4.2.3.11 `int g15r_loadWbmpSplash (g15canvas * canvas, char * filename)`

Draw a splash screen from 160x43 wbmp file.

wbmp splash screen loader - assumes image is 160x43

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>filename</i>	A string holding the path to the wbmp to be displayed.

Definition at line 387 of file pixel.c.

References `g15canvas::buffer`, `G15_BUFFER_LEN`, and `g15r_loadWbmpToBuf()`.

```

388 {
389     int width=0, height=0;
390     char *buf;
391
392     buf = g15r_loadWbmpToBuf(filename,
393                             &width,
394                             &height);
395
396     memcpy (canvas->buffer, buf, G15_BUFFER_LEN);
397     return 0;
398 }
```

4.2.3.12 `char* g15r_loadWbmpToBuf (char * filename, int * img_width, int * img_height)`

Load a wbmp file into a buffer.

basic wbmp loader - loads a wbmp image into a buffer.

Parameters

<i>filename</i>	A string holding the path to the wbmp to be loaded.
<i>img_width</i>	A pointer to an int that will hold the image width on return.
<i>img_height</i>	A pointer to an int that will hold the image height on return.

Definition at line 469 of file pixel.c.

References `BYTE_SIZE`.

Referenced by `g15r_loadWbmpSplash()`.

```

470 {
471     int wbmp_fd;
472     int retval;
473     int x,y,val;
474     char *buf;
475     unsigned int buflen,header=4;
```

```

476     unsigned char headerbytes[5];
477     unsigned int pixel_offset = 0;
478     unsigned int byte_offset, bit_offset;
479
480     wbmp_fd=open(filename,O_RDONLY);
481     if(!wbmp_fd){
482         return NULL;
483     }
484
485     retval=read(wbmp_fd,headerbytes,5);
486
487     if(retval){
488         if (headerbytes[2] & 1) {
489             *img_width = ((unsigned char)headerbytes[2] ^ 1) | (unsigned char)headerbytes[3];
490             *img_height = headerbytes[4];
491             header = 5;
492         } else {
493             *img_width = headerbytes[2];
494             *img_height = headerbytes[3];
495         }
496
497         int byte_width = *img_width / 8;
498         if (*img_width %8)
499             byte_width++;
500
501         buflen = byte_width * (*img_height);
502
503         buf = (char *)malloc (buflen);
504         if (buf == NULL)
505             return NULL;
506
507         if (header == 4)
508             buf[0]=headerbytes[4];
509
510         retval=read(wbmp_fd,buf+(5-header),buflen);
511
512         close(wbmp_fd);
513     }
514
515     /* now invert the image */
516     for (y = 0; y < *img_height; y++)
517         for (x = 0; x < *img_width; x++)
518             {
519                 pixel_offset = y * (*img_width) + x;
520                 byte_offset = pixel_offset / BYTE_SIZE;
521                 bit_offset = 7 - (pixel_offset % BYTE_SIZE);
522
523                 val = (buf[byte_offset] & (1 << bit_offset)) >> bit_offset;
524
525                 if (!val)
526                     buf[byte_offset] = buf[byte_offset] | 1 << bit_offset;
527                 else
528                     buf[byte_offset] = buf[byte_offset] & ~(1 << bit_offset);
529             }
530
531     return buf;
532 }

```

4.2.3.13 void g15r_pixelBox (g15canvas* canvas, int x1, int y1, int x2, int y2, int color, int thick, int fill)

Draws a box bounded by (x1, y1) and (x2, y2)

Draws a box around the area bounded by (x1, y1) and (x2, y2).

The box will be filled if fill != 0 and the sides will be thick pixels wide.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of the box.
<i>y1</i>	Defines uppermost bound of the box.
<i>x2</i>	Defines rightmost bound of the box.
<i>y2</i>	Defines bottommost bound of the box.

<i>color</i>	Lines defining the box will be drawn this color.
<i>thick</i>	Lines defining the box will be this many pixels thick.
<i>fill</i>	The box will be filled with color if fill != 0.

Definition at line 163 of file pixel.c.

References g15r_drawLine(), and g15r_setPixel().

Referenced by g15r_drawBar(), and g15r_drawBigNum().

```

165 {
166     int i = 0;
167     for (i = 0; i < thick; ++i)
168     {
169         g15r_drawLine (canvas, x1, y1, x2, y1, color);    /* Top */
170         g15r_drawLine (canvas, x1, y1, x1, y2, color);    /* Left */
171         g15r_drawLine (canvas, x2, y1, x2, y2, color);    /* Right */
172         g15r_drawLine (canvas, x1, y2, x2, y2, color);    /* Bottom */
173         x1++;
174         y1++;
175         x2--;
176         y2--;
177     }
178
179     int x = 0, y = 0;
180
181     if (fill)
182     {
183         for (x = x1; x <= x2; ++x)
184             for (y = y1; y <= y2; ++y)
185                 g15r_setPixel (canvas, x, y, color);
186     }
187
188 }
```

4.2.3.14 void g15r_pixelOverlay (g15canvas * canvas, int x1, int y1, int width, int height, short colormap[])

Overlays a bitmap of size width x height starting at (x1, y1)

A 1-bit bitmap defined in colormap[] is drawn to the canvas with an upper left corner at (x1, y1) and a lower right corner at (x1+width, y1+height).

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines the leftmost bound of the area to be drawn.
<i>y1</i>	Defines the uppermost bound of the area to be drawn.
<i>width</i>	Defines the width of the bitmap to be drawn.
<i>height</i>	Defines the height of the bitmap to be drawn.
<i>colormap</i>	An array containing width*height entries of value 0 for pixel off or != 0 for pixel on.

Definition at line 74 of file pixel.c.

References G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

```

76 {
77     int i = 0;
78
79     for (i = 0; i < (width * height); ++i)
80     {
81         int color = (colormap[i] ? G15_COLOR_BLACK : G15_COLOR_WHITE);
82         int x = x1 + i % width;
83         int y = y1 + i / width;
84         g15r_setPixel (canvas, x, y, color);
85     }
86 }
```

4.2.3.15 void g15r_pixelReverseFill (g15canvas * canvas, int x1, int y1, int x2, int y2, int fill, int color)

Fills an area bounded by (x1, y1) and (x2, y2)

The area with an upper left corner at (x1, y1) and lower right corner at (x2, y2) will be filled with color if fill>0 or the current contents of the area will be reversed if fill==0.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of area to be filled.
<i>y1</i>	Defines uppermost bound of area to be filled.
<i>x2</i>	Defines rightmost bound of area to be filled.
<i>y2</i>	Defines bottommost bound of area to be filled.
<i>fill</i>	Area will be filled with color if fill != 0, else contents of area will have color values reversed.
<i>color</i>	If fill != 0, then area will be filled if color == 1 and emptied if color == 0.

Definition at line 45 of file pixel.c.

References g15r_getPixel(), and g15r_setPixel().

```

47 {
48     int x = 0;
49     int y = 0;
50
51     for (x = x1; x <= x2; ++x)
52     {
53         for (y = y1; y <= y2; ++y)
54         {
55             if (!fill)
56                 color = !g15r_getPixel (canvas, x, y);
57             g15r_setPixel (canvas, x, y, color);
58         }
59     }
60 }
```

4.2.3.16 void g15r_renderCharacterLarge (g15canvas * *canvas*, int *x*, int *y*, unsigned char *character*, unsigned int *sx*, unsigned int *sy*)

Renders a character in the large font at (x, y)

Definition at line 22 of file text.c.

References fontdata_8x8, G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

Referenced by g15r_renderString().

```

25 {
26     int helper = character * 8;    /* for our font which is 8x8 */
27
28     int top_left_pixel_x = sx + col * (8);    /* 1 pixel spacing */
29     int top_left_pixel_y = sy + row * (8);    /* once again 1 pixel spacing */
30
31     int x, y;
32     for (y = 0; y < 8; ++y)
33     {
34         for (x = 0; x < 8; ++x)
35         {
36             char font_entry = fontdata_8x8[helper + y];
37
38             if (font_entry & 1 << (7 - x))
39                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
40                               G15_COLOR_BLACK);
41             else
42                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
43                               G15_COLOR_WHITE);
44         }
45     }
46 }
47 }
```

4.2.3.17 void g15r_renderCharacterMedium (g15canvas * *canvas*, int *x*, int *y*, unsigned char *character*, unsigned int *sx*, unsigned int *sy*)

Renders a character in the meduim font at (x, y)

Definition at line 50 of file text.c.

References fontdata_7x5, G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

Referenced by g15r_renderString().

```

53 {
54     int helper = character * 7 * 5;          /* for our font which is 6x4 */
55
56     int top_left_pixel_x = sx + col * (5);    /* 1 pixel spacing */
57     int top_left_pixel_y = sy + row * (7);    /* once again 1 pixel spacing */
58
59     int x, y;
60     for (y = 0; y < 7; ++y)
61     {
62         for (x = 0; x < 5; ++x)
63         {
64             char font_entry = fontdata_7x5[helper + y * 5 + x];
65             if (font_entry)
66                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
67                               G15_COLOR_BLACK);
68             else
69                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
70                               G15_COLOR_WHITE);
71         }
72     }
73 }
74 }
```

4.2.3.18 void g15r_renderCharacterSmall (g15canvas * canvas, int x, int y, unsigned char character, unsigned int sx, unsigned int sy)

Renders a character in the small font at (x, y)

Definition at line 77 of file text.c.

References fontdata_6x4, G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

Referenced by g15r_renderString().

```

80 {
81     int helper = character * 6 * 4;          /* for our font which is 6x4 */
82
83     int top_left_pixel_x = sx + col * (4);    /* 1 pixel spacing */
84     int top_left_pixel_y = sy + row * (6);    /* once again 1 pixel spacing */
85
86     int x, y;
87     for (y = 0; y < 6; ++y)
88     {
89         for (x = 0; x < 4; ++x)
90         {
91             char font_entry = fontdata_6x4[helper + y * 4 + x];
92             if (font_entry)
93                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
94                               G15_COLOR_BLACK);
95             else
96                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
97                               G15_COLOR_WHITE);
98         }
99     }
100 }
101 }
```

4.2.3.19 void g15r_renderString (g15canvas * canvas, unsigned char stringWith[], int row, int size, unsigned int sx, unsigned int sy)

Renders a string with font size in row.

Definition at line 104 of file text.c.

References G15_TEXT_LARGE, G15_TEXT_MED, G15_TEXT_SMALL, g15r_renderCharacterLarge(), g15r_renderCharacterMedium(), and g15r_renderCharacterSmall().

```

106 {
107
```

```

108  int i = 0;
109  for (i; stringOut[i] != NULL; ++i)
110  {
111      switch (size)
112      {
113          case G15_TEXT_SMALL:
114          {
115              g15r_renderCharacterSmall (canvas, i, row, stringOut[i], sx, sy);
116              break;
117          }
118          case G15_TEXT_MED:
119          {
120              g15r_renderCharacterMedium (canvas, i, row, stringOut[i], sx, sy);
121              break;
122          }
123          case G15_TEXT_LARGE:
124          {
125              g15r_renderCharacterLarge (canvas, i, row, stringOut[i], sx, sy);
126              break;
127          }
128          default:
129              break;
130      }
131  }
132
133 }

```

4.2.3.20 void g15r_setPixel (g15canvas * canvas, unsigned int x, unsigned int y, int val)

Sets the value of the pixel at (x, y)

Sets the value of the pixel at (x, y)

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x</i>	X offset for pixel to be set.
<i>y</i>	Y offset for pixel to be set.
<i>val</i>	Value to which pixel should be set.

Definition at line 50 of file screen.c.

References g15canvas::buffer, BYTE_SIZE, G15_LCD_HEIGHT, G15_LCD_WIDTH, g15r_getPixel(), g15canvas::mode_reverse, and g15canvas::mode_xor.

Referenced by draw_ttf_char(), g15r_drawCircle(), g15r_drawIcon(), g15r_drawLine(), g15r_drawRoundBox(), g15r_drawSprite(), g15r_pixelBox(), g15r_pixelOverlay(), g15r_pixelReverseFill(), g15r_renderCharacterLarge(), g15r_renderCharacterMedium(), and g15r_renderCharacterSmall().

```

51 {
52     if (x >= G15_LCD_WIDTH || y >= G15_LCD_HEIGHT)
53         return;
54
55     unsigned int pixel_offset = y * G15_LCD_WIDTH + x;
56     unsigned int byte_offset = pixel_offset / BYTE_SIZE;
57     unsigned int bit_offset = 7 - (pixel_offset % BYTE_SIZE);
58
59     if (canvas->mode_xor)
60         val ^= g15r_getPixel (canvas, x, y);
61     if (canvas->mode_reverse)
62         val = !val;
63
64     if (val)
65         canvas->buffer[byte_offset] =
66             canvas->buffer[byte_offset] | 1 << bit_offset;
67     else
68         canvas->buffer[byte_offset] =
69             canvas->buffer[byte_offset] & ~(1 << bit_offset);
70
71 }

```

4.2.3.21 void g15r_ttfLoad (g15canvas * canvas, char * fontname, int fontsize, int face_num)

Loads a font through the FreeType2 library.

Load a font for use with FreeType2 font support

Parameters

<i>canvas</i>	A pointer to a g15canvas (p.5) struct in which the buffer to be operated on is found.
<i>fontname</i>	Absolute pathname to font file to be loaded.
<i>fontsize</i>	Size in points for font to be loaded.
<i>face_num</i>	Slot into which font face will be loaded.

Definition at line 145 of file text.c.

References g15canvas::ftLib, G15_MAX_FACE, g15canvas::ttf_face, and g15canvas::ttf_fontsize.

```

146 {
147     int errcode = 0;
148
149     if (face_num < 0)
150         face_num = 0;
151     if (face_num > G15_MAX_FACE)
152         face_num = G15_MAX_FACE;
153
154     if (canvas->ttf_fontsize[face_num])
155         FT_Done_Face (canvas->ttf_face[face_num][0]);          /* destroy the last face */
156
157     if (!canvas->ttf_fontsize[face_num] && !fontsize)
158         canvas->ttf_fontsize[face_num] = 10;
159     else
160         canvas->ttf_fontsize[face_num] = fontsize;
161
162     errcode =
163         FT_New_Face (canvas->ftLib, fontname, 0, &canvas->ttf_face[face_num][0]);
164     if (errcode)
165     {
166         canvas->ttf_fontsize[face_num] = 0;
167     }
168     else
169     {
170         if (canvas->ttf_fontsize[face_num]
171             && FT_IS_SCALABLE (canvas->ttf_face[face_num][0]))
172             errcode =
173                 FT_Set_Char_Size (canvas->ttf_face[face_num][0], 0,
174                                 canvas->ttf_fontsize[face_num] * 64, 90, 0);
175     }
176 }
```

4.2.3.22 void g15r_ttfPrint (g15canvas * canvas, int x, int y, int fontsize, int face_num, int color, int center, char * print_string)

Prints a string in a given font.

Render a string with a FreeType2 font

Parameters

<i>canvas</i>	A pointer to a g15canvas (p.5) struct in which the buffer to be operated on is found.
<i>x</i>	initial x position for string.
<i>y</i>	initial y position for string.
<i>fontsize</i>	Size of string in points.
<i>face_num</i>	Font to be used is loaded in this slot.
<i>color</i>	Text will be drawn this color.
<i>center</i>	Text will be centered if center == 1 and right justified if center == 2.
<i>print_string</i>	Pointer to the string to be printed.

Definition at line 283 of file text.c.

References calc_ttf_centering(), calc_ttf_right_justify(), calc_ttf_true_ypos(), draw_ttf_str(), g15canvas::ttf_face, and g15canvas::ttf_fontsize.

```

285 {
286     int errcode = 0;
287
288     if (canvas->ttf_fontsize[face_num])
289     {
290         if (fontsize > 0 && FT_IS_SCALABLE (canvas->ttf_face[face_num][0]))
```

```

291     {
292         canvas->ttf_fontsize[face_num] = fontsize;
293         int errcode =
294             FT_Set_Pixel_Sizes (canvas->ttf_face[face_num][0], 0,
295                               canvas->ttf_fontsize[face_num]);
296         if (errcode)
297             printf ("Trouble setting the Glyph size!\n");
298     }
299     y =
300         calc_ttf_true_ypos (canvas->ttf_face[face_num][0], y,
301                             canvas->ttf_fontsize[face_num]);
302     if (center == 1)
303         x = calc_ttf_centering (canvas->ttf_face[face_num][0], print_string);
304     else if (center == 2)
305         x = calc_ttf_right_justify (canvas->ttf_face[face_num][0], print_string);
306     draw_ttf_str (canvas, print_string, x, y, color,
307                  canvas->ttf_face[face_num][0]);
308 }
309 }

```

4.2.4 Variable Documentation

4.2.4.1 unsigned char fontdata_6x4[]

Font data for the small (6x4) font.

Referenced by `g15r_renderCharacterSmall()`.

4.2.4.2 unsigned char fontdata_7x5[]

Font data for the medium (7x5) font.

Referenced by `g15r_renderCharacterMedium()`.

4.2.4.3 unsigned char fontdata_8x8[]

Font data for the large (8x8) font.

Referenced by `g15r_renderCharacterLarge()`.

4.3 pixel.c File Reference

```

#include <fcntl.h>
#include "libg15render.h"

```

Functions

- void **g15r_drawBar** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int color, int num, int max, int type)
Draws a completion bar.
- void **g15r_drawBigNum** (**g15canvas** *canvas, unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int color, int num)
Draw a large number.
- void **g15r_drawCircle** (**g15canvas** *canvas, int x, int y, int r, int fill, int color)
Draws a circle centered at (x, y) with a radius of r.
- void **g15r_drawIcon** (**g15canvas** *canvas, char *buf, int my_x, int my_y, int width, int height)
Draw an icon to the screen from a wbmp buffer.
- void **g15r_drawLine** (**g15canvas** *canvas, int px1, int py1, int px2, int py2, const int color)
Draws a line from (px1, py1) to (px2, py2)

- void **g15r_drawRoundBox** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int fill, int color)
Draws a box with rounded corners bounded by (x1, y1) and (x2, y2)
- void **g15r_drawSprite** (**g15canvas** *canvas, char *buf, int my_x, int my_y, int width, int height, int start_x, int start_y, int total_width)
Draw a sprite to the screen from a wbmp buffer.
- int **g15r_loadWbmpSplash** (**g15canvas** *canvas, char *filename)
Draw a splash screen from 160x43 wbmp file.
- char * **g15r_loadWbmpToBuf** (char *filename, int *img_width, int *img_height)
Load a wbmp file into a buffer.
- void **g15r_pixelBox** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int color, int thick, int fill)
Draws a box bounded by (x1, y1) and (x2, y2)
- void **g15r_pixelOverlay** (**g15canvas** *canvas, int x1, int y1, int width, int height, short colormap[])
Overlays a bitmap of size width x height starting at (x1, y1)
- void **g15r_pixelReverseFill** (**g15canvas** *canvas, int x1, int y1, int x2, int y2, int fill, int color)
Fills an area bounded by (x1, y1) and (x2, y2)
- void **swap** (int *x, int *y)

4.3.1 Function Documentation

4.3.1.1 void g15r_drawBar (g15canvas * canvas, int x1, int y1, int x2, int y2, int color, int num, int max, int type)

Draws a completion bar.

Given a maximum value, and a value between 0 and that maximum value, calculate and draw a bar showing that percentage.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of the bar.
<i>y1</i>	Defines uppermost bound of the bar.
<i>x2</i>	Defines rightmost bound of the bar.
<i>y2</i>	Defines bottommost bound of the bar.
<i>color</i>	The bar will be drawn this color.
<i>num</i>	Number of units relative to max filled.
<i>max</i>	Number of units equal to 100% filled.
<i>type</i>	Type of bar. 1=solid bar, 2=solid bar with border, 3 = solid bar with l-frame.

Definition at line 337 of file pixel.c.

References g15r_drawLine(), and g15r_pixelBox().

```

339 {
340     float len, length;
341     int x;
342     if (max == 0)
343         return;
344     if (num > max)
345         num = max;
346
347     if (type == 2)
348     {
349         y1 += 2;
350         y2 -= 2;
351         x1 += 2;
352         x2 -= 2;
353     }
354
355     len = ((float) max / (float) num);
356     length = (x2 - x1) / len;
357
358     if (type == 1)
359     {
360         g15r_pixelBox (canvas, x1, y1 - type, x2, y2 + type, color ^ 1, 1, 1);

```

```

361     gl5r_pixelBox (canvas, x1, y1 - type, x2, y2 + type, color, 1, 0);
362 }
363 else if (type == 2)
364 {
365     gl5r_pixelBox (canvas, x1 - 2, y1 - type, x2 + 2, y2 + type, color ^ 1,
366                   1, 1);
367     gl5r_pixelBox (canvas, x1 - 2, y1 - type, x2 + 2, y2 + type, color, 1,
368                   0);
369 }
370 else if (type == 3)
371 {
372     gl5r_drawLine (canvas, x1, y1 - type, x1, y2 + type, color);
373     gl5r_drawLine (canvas, x2, y1 - type, x2, y2 + type, color);
374     gl5r_drawLine (canvas, x1, y1 + ((y2 - y1) / 2), x2,
375                   y1 + ((y2 - y1) / 2), color);
376 }
377 gl5r_pixelBox (canvas, x1, y1, (int) ceil (x1 + length), y2, color, 1, 1);
378 }

```

4.3.1.2 void gl5r_drawBigNum (g15canvas * canvas, unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int color, int num)

Draw a large number.

Draw a large number to a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of the number.
<i>y1</i>	Defines uppermost bound of the number.
<i>x2</i>	Defines rightmost bound of the number.
<i>y2</i>	Defines bottommost bound of the number.
<i>num</i>	The number to be drawn.

Definition at line 545 of file pixel.c.

References gl5r_pixelBox().

```

546 {
547     x1 += 2;
548     x2 -= 2;
549
550     switch(num) {
551         case 0:
552             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
553             gl5r_pixelBox (canvas, x1 + 5, y1 + 5, x2 - 5, y2 - 6, 1 - color, 1, 1);
554             break;
555         case 1:
556             gl5r_pixelBox (canvas, x2 - 5, y1, x2, y2 , color, 1, 1);
557             gl5r_pixelBox (canvas, x1, y1, x2 - 5, y2, 1 - color, 1, 1);
558             break;
559         case 2:
560             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
561             gl5r_pixelBox (canvas, x1, y1 + 5, x2 - 5, y1 + ((y2 / 2) - 3), 1 - color, 1, 1);
562             gl5r_pixelBox (canvas, x1 + 5, y1 + ((y2 / 2) + 3), x2 , y2 - 6, 1 - color, 1, 1);
563             break;
564         case 3:
565             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
566             gl5r_pixelBox (canvas, x1, y1 + 5, x2 - 5, y1 + ((y2 / 2) - 3), 1 - color, 1, 1);
567             gl5r_pixelBox (canvas, x1, y1 + ((y2 / 2) + 3), x2 - 5 , y2 - 6, 1 - color, 1, 1);
568             break;
569         case 4:
570             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
571             gl5r_pixelBox (canvas, x1, y1 + ((y2 / 2) + 3), x2 - 5, y2, 1 - color, 1, 1);
572             gl5r_pixelBox (canvas, x1 + 5, y1, x2 - 5 , y1 + ((y2 / 2) - 3), 1 - color, 1, 1);
573             break;
574         case 5:
575             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
576             gl5r_pixelBox (canvas, x1 + 5, y1 + 5, x2 , y1 + ((y2 / 2) - 3), 1 - color, 1, 1);
577             gl5r_pixelBox (canvas, x1, y1 + ((y2 / 2) + 3), x2 - 5 , y2 - 6, 1 - color, 1, 1);
578             break;
579         case 6:
580             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
581             gl5r_pixelBox (canvas, x1 + 5, y1 + 5, x2 , y1 + ((y2 / 2) - 3), 1 - color, 1, 1);
582             gl5r_pixelBox (canvas, x1 + 5, y1 + ((y2 / 2) + 3), x2 - 5 , y2 - 6, 1 - color, 1, 1);
583             break;

```

```

584         case 7:
585             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
586             gl5r_pixelBox (canvas, x1, y1+5, x2 -5, y2, 1 - color, 1, 1);
587             break;
588         case 8:
589             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
590             gl5r_pixelBox (canvas, x1+5, y1+5, x2-5 , y1+((y2/2)-3), 1 - color, 1, 1);
591             gl5r_pixelBox (canvas, x1+5, y1+((y2/2)+3), x2-5 , y2-6, 1 - color, 1, 1);
592             break;
593         case 9:
594             gl5r_pixelBox (canvas, x1, y1, x2, y2 , color, 1, 1);
595             gl5r_pixelBox (canvas, x1+5, y1+5, x2-5 , y1+((y2/2)-3), 1 - color, 1, 1);
596             gl5r_pixelBox (canvas, x1, y1+((y2/2)+3), x2-5 , y2, 1 - color, 1, 1);
597             break;
598         case 10:
599             gl5r_pixelBox (canvas, x2-5, y1+5, x2, y1+10 , color, 1, 1);
600             gl5r_pixelBox (canvas, x2-5, y2-10, x2, y2-5 , color, 1, 1);
601             break;
602         case 11:
603             gl5r_pixelBox (canvas, x1, y1+((y2/2)-2), x2, y1+((y2/2)+2), color, 1, 1);
604             break;
605         case 12:
606             gl5r_pixelBox (canvas, x2-5, y2-5, x2, y2 , color, 1, 1);
607             break;
608     }
609 }

```

4.3.1.3 void gl5r_drawCircle (g15canvas * canvas, int x, int y, int r, int fill, int color)

Draws a circle centered at (x, y) with a radius of r.

Draws a circle centered at (x, y) with a radius of r.

The circle will be filled if fill != 0.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p.5) struct in which the buffer to be operated on is found.
<i>x</i>	Defines horizontal center of the circle.
<i>y</i>	Defines vertical center of circle.
<i>r</i>	Defines radius of circle.
<i>fill</i>	The circle will be filled with color if fill != 0.
<i>color</i>	Lines defining the circle will be drawn this color.

Definition at line 203 of file pixel.c.

References gl5r_drawLine(), and gl5r_setPixel().

```

204 {
205     int xx, yy, dd;
206
207     xx = 0;
208     yy = r;
209     dd = 2 * (1 - r);
210
211     while (yy >= 0)
212     {
213         if (!fill)
214         {
215             gl5r_setPixel (canvas, x + xx, y - yy, color);
216             gl5r_setPixel (canvas, x + xx, y + yy, color);
217             gl5r_setPixel (canvas, x - xx, y - yy, color);
218             gl5r_setPixel (canvas, x - xx, y + yy, color);
219         }
220         else
221         {
222             gl5r_drawLine (canvas, x - xx, y - yy, x + xx, y - yy, color);
223             gl5r_drawLine (canvas, x - xx, y + yy, x + xx, y + yy, color);
224         }
225         if (dd + yy > 0)
226         {
227             yy--;
228             dd = dd - (2 * yy + 1);
229         }
230         if (xx > dd)
231         {
232             xx++;
233             dd = dd + (2 * xx + 1);

```

```

234     }
235 }
236 }

```

4.3.1.4 void g15r_drawIcon (g15canvas * canvas, char * buf, int my_x, int my_y, int width, int height)

Draw an icon to the screen from a wbmp buffer.

Draw an icon to a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated in is found.
<i>buf</i>	A pointer to the buffer holding the icon to be displayed.
<i>my_x</i>	Leftmost boundary of image.
<i>my_y</i>	Topmost boundary of image.
<i>width</i>	Width of the image in buf.
<i>height</i>	Height of the image in buf.

Definition at line 411 of file pixel.c.

References `BYTE_SIZE`, and `g15r_setPixel()`.

```

412 {
413     int y,x,val;
414     unsigned int pixel_offset = 0;
415     unsigned int byte_offset, bit_offset;
416
417     for (y=0; y < height - 1; y++)
418         for (x=0; x < width - 1; x++)
419         {
420             pixel_offset = y * width + x;
421             byte_offset = pixel_offset / BYTE_SIZE;
422             bit_offset = 7 - (pixel_offset % BYTE_SIZE);
423
424             val = (buf[byte_offset] & (1 << bit_offset)) >> bit_offset;
425             g15r_setPixel (canvas, x + my_x, y + my_y, val);
426         }
427 }

```

4.3.1.5 void g15r_drawLine (g15canvas * canvas, int px1, int py1, int px2, int py2, const int color)

Draws a line from (px1, py1) to (px2, py2)

A line of color is drawn from (px1, py1) to (px2, py2).

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>px1</i>	X component of point 1.
<i>py1</i>	Y component of point 1.
<i>px2</i>	X component of point 2.
<i>py2</i>	Y component of point 2.
<i>color</i>	Line will be drawn this color.

Definition at line 99 of file pixel.c.

References `g15r_setPixel()`, and `swap()`.

Referenced by `g15r_drawBar()`, `g15r_drawCircle()`, `g15r_drawRoundBox()`, and `g15r_pixelBox()`.

```

101 {
102     /*
103      * Bresenham's Line Algorithm
104      * http://en.wikipedia.org/wiki/Bresenham's_algorithm
105      */
106
107     int steep = 0;

```

```

108
109  if (abs (py2 - py1) > abs (px2 - px1))
110      steep = 1;
111
112  if (steep)
113  {
114      swap (&px1, &py1);
115      swap (&px2, &py2);
116  }
117
118  if (px1 > px2)
119  {
120      swap (&px1, &px2);
121      swap (&py1, &py2);
122  }
123
124  int dx = px2 - px1;
125  int dy = abs (py2 - py1);
126
127  int error = 0;
128  int y = py1;
129  int ystep = (py1 < py2) ? 1 : -1;
130  int x = 0;
131
132  for (x = px1; x <= px2; ++x)
133  {
134      if (steep)
135          gl5r_setPixel (canvas, y, x, color);
136      else
137          gl5r_setPixel (canvas, x, y, color);
138
139      error += dy;
140      if (2 * error >= dx)
141      {
142          y += ystep;
143          error -= dx;
144      }
145  }
146 }

```

4.3.1.6 void gl5r_drawRoundBox (g15canvas * canvas, int x1, int y1, int x2, int y2, int fill, int color)

Draws a box with rounded corners bounded by (x1, y1) and (x2, y2)

Draws a rounded box around the area bounded by (x1, y1) and (x2, y2).

The box will be filled if fill != 0.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of the box.
<i>y1</i>	Defines uppermost bound of the box.
<i>x2</i>	Defines rightmost bound of the box.
<i>y2</i>	Defines bottommost bound of the box.
<i>fill</i>	The box will be filled with color if fill != 0.
<i>color</i>	Lines defining the box will be drawn this color.

Definition at line 252 of file pixel.c.

References G15_COLOR_BLACK, G15_COLOR_WHITE, gl5r_drawLine(), and gl5r_setPixel().

```

254 {
255     int y, shave = 3;
256
257     if (shave > (x2 - x1) / 2)
258         shave = (x2 - x1) / 2;
259     if (shave > (y2 - y1) / 2)
260         shave = (y2 - y1) / 2;
261
262     if ((x1 != x2) && (y1 != y2))
263     {
264         if (fill)
265         {
266             gl5r_drawLine (canvas, x1 + shave, y1, x2 - shave, y1, color);
267             for (y = y1 + 1; y < y1 + shave; y++)
268                 gl5r_drawLine (canvas, x1 + 1, y, x2 - 1, y, color);

```



```

269     for (y = y1 + shave; y <= y2 - shave; y++)
270         g15r_drawLine (canvas, x1, y, x2, y, color);
271     for (y = y2 - shave + 1; y < y2; y++)
272         g15r_drawLine (canvas, x1 + 1, y, x2 - 1, y, color);
273     g15r_drawLine (canvas, x1 + shave, y2, x2 - shave, y2, color);
274     if (shave == 4)
275     {
276         g15r_setPixel (canvas, x1 + 1, y1 + 1,
277             color ==
278             G15_COLOR_WHITE ? G15_COLOR_BLACK :
279             G15_COLOR_WHITE);
280         g15r_setPixel (canvas, x1 + 1, y2 - 1,
281             color ==
282             G15_COLOR_WHITE ? G15_COLOR_BLACK :
283             G15_COLOR_WHITE);
284         g15r_setPixel (canvas, x2 - 1, y1 + 1,
285             color ==
286             G15_COLOR_WHITE ? G15_COLOR_BLACK :
287             G15_COLOR_WHITE);
288         g15r_setPixel (canvas, x2 - 1, y2 - 1,
289             color ==
290             G15_COLOR_WHITE ? G15_COLOR_BLACK :
291             G15_COLOR_WHITE);
292     }
293 }
294 else
295 {
296     g15r_drawLine (canvas, x1 + shave, y1, x2 - shave, y1, color);
297     g15r_drawLine (canvas, x1, y1 + shave, x1, y2 - shave, color);
298     g15r_drawLine (canvas, x2, y1 + shave, x2, y2 - shave, color);
299     g15r_drawLine (canvas, x1 + shave, y2, x2 - shave, y2, color);
300     if (shave > 1)
301     {
302         g15r_drawLine (canvas, x1 + 1, y1 + 1, x1 + shave - 1, y1 + 1,
303             color);
304         g15r_drawLine (canvas, x2 - shave + 1, y1 + 1, x2 - 1, y1 + 1,
305             color);
306         g15r_drawLine (canvas, x1 + 1, y2 - 1, x1 + shave - 1, y2 - 1,
307             color);
308         g15r_drawLine (canvas, x2 - shave + 1, y2 - 1, x2 - 1, y2 - 1,
309             color);
310         g15r_drawLine (canvas, x1 + 1, y1 + 1, x1 + 1, y1 + shave - 1,
311             color);
312         g15r_drawLine (canvas, x1 + 1, y2 - 1, x1 + 1, y2 - shave + 1,
313             color);
314         g15r_drawLine (canvas, x2 - 1, y1 + 1, x2 - 1, y1 + shave - 1,
315             color);
316         g15r_drawLine (canvas, x2 - 1, y2 - 1, x2 - 1, y2 - shave + 1,
317             color);
318     }
319 }
320 }
321 }

```

4.3.1.7 void g15r_drawSprite (g15canvas * canvas, char * buf, int my_x, int my_y, int width, int height, int start_x, int start_y, int total_width)

Draw a sprite to the screen from a wbmp buffer.

Draw a sprite to a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated in is found.
<i>buf</i>	A pointer to the buffer holding a set of sprites.
<i>my_x</i>	Leftmost boundary of image.
<i>my_y</i>	Topmost boundary of image.
<i>width</i>	Width of the sprite.
<i>height</i>	Height of the sprite.
<i>start_x</i>	X offset for reading sprite from buf.

<i>start_y</i>	Y offset for reading sprite from buf.
<i>total_width</i>	Width of the set of sprites held in buf.

Definition at line 443 of file pixel.c.

References `BYTE_SIZE`, and `g15r_setPixel()`.

```

444 {
445     int y,x,val;
446     unsigned int pixel_offset = 0;
447     unsigned int byte_offset, bit_offset;
448
449     for (y=0; y < height - 1; y++)
450         for (x=0; x < width - 1; x++)
451         {
452             pixel_offset = (y + start_y) * total_width + (x + start_x);
453             byte_offset = pixel_offset / BYTE_SIZE;
454             bit_offset = 7 - (pixel_offset % BYTE_SIZE);
455
456             val = (buf[byte_offset] & (1 << bit_offset)) >> bit_offset;
457             g15r_setPixel (canvas, x + my_x, y + my_y, val);
458         }
459 }
```

4.3.1.8 `int g15r_loadWbmpSplash (g15canvas* canvas, char * filename)`

Draw a splash screen from 160x43 wbmp file.

wbmp splash screen loader - assumes image is 160x43

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>filename</i>	A string holding the path to the wbmp to be displayed.

Definition at line 387 of file pixel.c.

References `g15canvas::buffer`, `G15_BUFFER_LEN`, and `g15r_loadWbmpToBuf()`.

```

388 {
389     int width=0, height=0;
390     char *buf;
391
392     buf = g15r_loadWbmpToBuf(filename,
393                             &width,
394                             &height);
395
396     memcpy (canvas->buffer, buf, G15_BUFFER_LEN);
397     return 0;
398 }
```

4.3.1.9 `char* g15r_loadWbmpToBuf (char * filename, int * img_width, int * img_height)`

Load a wbmp file into a buffer.

basic wbmp loader - loads a wbmp image into a buffer.

Parameters

<i>filename</i>	A string holding the path to the wbmp to be loaded.
<i>img_width</i>	A pointer to an int that will hold the image width on return.
<i>img_height</i>	A pointer to an int that will hold the image height on return.

Definition at line 469 of file pixel.c.

References `BYTE_SIZE`.

Referenced by `g15r_loadWbmpSplash()`.

```

470 {
471     int wbmp_fd;
472     int retval;
473     int x,y,val;
474     char *buf;
475     unsigned int buflen,header=4;
476     unsigned char headerbytes[5];
477     unsigned int pixel_offset = 0;
478     unsigned int byte_offset, bit_offset;
479
480     wbmp_fd=open(filename,O_RDONLY);
481     if(!wbmp_fd){
482         return NULL;
483     }
484
485     retval=read(wbmp_fd,headerbytes,5);
486
487     if(retval){
488         if (headerbytes[2] & 1) {
489             *img_width = ((unsigned char)headerbytes[2] ^ 1) | (unsigned char)headerbytes[3];
490             *img_height = headerbytes[4];
491             header = 5;
492         } else {
493             *img_width = headerbytes[2];
494             *img_height = headerbytes[3];
495         }
496
497         int byte_width = *img_width / 8;
498         if (*img_width %8)
499             byte_width++;
500
501         buflen = byte_width * (*img_height);
502
503         buf = (char *)malloc (buflen);
504         if (buf == NULL)
505             return NULL;
506
507         if (header == 4)
508             buf[0]=headerbytes[4];
509
510         retval=read(wbmp_fd,buf+(5-header),buflen);
511
512         close(wbmp_fd);
513     }
514
515     /* now invert the image */
516     for (y = 0; y < *img_height; y++)
517         for (x = 0; x < *img_width; x++)
518             {
519                 pixel_offset = y * (*img_width) + x;
520                 byte_offset = pixel_offset / BYTE_SIZE;
521                 bit_offset = 7 - (pixel_offset % BYTE_SIZE);
522
523                 val = (buf[byte_offset] & (1 << bit_offset)) >> bit_offset;
524
525                 if (!val)
526                     buf[byte_offset] = buf[byte_offset] | 1 << bit_offset;
527                 else
528                     buf[byte_offset] = buf[byte_offset] & ~(1 << bit_offset);
529             }
530
531     return buf;
532 }

```

4.3.1.10 void g15r_pixelBox (g15canvas * canvas, int x1, int y1, int x2, int y2, int color, int thick, int fill)

Draws a box bounded by (x1, y1) and (x2, y2)

Draws a box around the area bounded by (x1, y1) and (x2, y2).

The box will be filled if fill != 0 and the sides will be thick pixels wide.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
---------------	---

<i>x1</i>	Defines leftmost bound of the box.
<i>y1</i>	Defines uppermost bound of the box.
<i>x2</i>	Defines rightmost bound of the box.
<i>y2</i>	Defines bottommost bound of the box.
<i>color</i>	Lines defining the box will be drawn this color.
<i>thick</i>	Lines defining the box will be this many pixels thick.
<i>fill</i>	The box will be filled with color if fill != 0.

Definition at line 163 of file pixel.c.

References g15r_drawLine(), and g15r_setPixel().

Referenced by g15r_drawBar(), and g15r_drawBigNum().

```

165 {
166     int i = 0;
167     for (i = 0; i < thick; ++i)
168     {
169         g15r_drawLine (canvas, x1, y1, x2, y1, color);    /* Top */
170         g15r_drawLine (canvas, x1, y1, x1, y2, color);    /* Left */
171         g15r_drawLine (canvas, x2, y1, x2, y2, color);    /* Right */
172         g15r_drawLine (canvas, x1, y2, x2, y2, color);    /* Bottom */
173         x1++;
174         y1++;
175         x2--;
176         y2--;
177     }
178
179     int x = 0, y = 0;
180
181     if (fill)
182     {
183         for (x = x1; x <= x2; ++x)
184             for (y = y1; y <= y2; ++y)
185                 g15r_setPixel (canvas, x, y, color);
186     }
187
188 }
```

4.3.1.11 void g15r_pixelOverlay (g15canvas * canvas, int x1, int y1, int width, int height, short colormap[])

Overlays a bitmap of size width x height starting at (x1, y1)

A 1-bit bitmap defined in colormap[] is drawn to the canvas with an upper left corner at (x1, y1) and a lower right corner at (x1+width, y1+height).

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines the leftmost bound of the area to be drawn.
<i>y1</i>	Defines the uppermost bound of the area to be drawn.
<i>width</i>	Defines the width of the bitmap to be drawn.
<i>height</i>	Defines the height of the bitmap to be drawn.
<i>colormap</i>	An array containing width*height entries of value 0 for pixel off or != 0 for pixel on.

Definition at line 74 of file pixel.c.

References G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

```

76 {
77     int i = 0;
78
79     for (i = 0; i < (width * height); ++i)
80     {
81         int color = (colormap[i] ? G15_COLOR_BLACK : G15_COLOR_WHITE);
82         int x = x1 + i % width;
83         int y = y1 + i / width;
84         g15r_setPixel (canvas, x, y, color);
85     }
86 }
```

4.3.1.12 void g15r_pixelReverseFill (g15canvas * canvas, int x1, int y1, int x2, int y2, int fill, int color)

Fills an area bounded by (x1, y1) and (x2, y2)

The area with an upper left corner at (x1, y1) and lower right corner at (x2, y2) will be filled with color if fill>0 or the current contents of the area will be reversed if fill==0.

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x1</i>	Defines leftmost bound of area to be filled.
<i>y1</i>	Defines uppermost bound of area to be filled.
<i>x2</i>	Defines rightmost bound of area to be filled.
<i>y2</i>	Defines bottommost bound of area to be filled.
<i>fill</i>	Area will be filled with color if fill != 0, else contents of area will have color values reversed.
<i>color</i>	If fill != 0, then area will be filled if color == 1 and emptied if color == 0.

Definition at line 45 of file pixel.c.

References g15r_getPixel(), and g15r_setPixel().

```

47 {
48     int x = 0;
49     int y = 0;
50
51     for (x = x1; x <= x2; ++x)
52     {
53         for (y = y1; y <= y2; ++y)
54         {
55             if (!fill)
56                 color = !g15r_getPixel (canvas, x, y);
57             g15r_setPixel (canvas, x, y, color);
58         }
59     }
60 }
```

4.3.1.13 void swap (int * x, int * y)

Definition at line 23 of file pixel.c.

Referenced by g15r_drawLine().

```

24 {
25     int tmp;
26
27     tmp = *x;
28     *x = *y;
29     *y = tmp;
30 }
```

4.4 screen.c File Reference

```
#include "libg15render.h"
```

Functions

- void **g15r_clearScreen** (g15canvas *canvas, int color)
Fills the screen with pixels of color.
- int **g15r_getPixel** (g15canvas *canvas, unsigned int x, unsigned int y)
Gets the value of the pixel at (x, y)
- void **g15r_initCanvas** (g15canvas *canvas)
Clears the canvas and resets the mode switches.

- void **g15r_setPixel** (**g15canvas** *canvas, unsigned int x, unsigned int y, int val)
Sets the value of the pixel at (x, y)

4.4.1 Function Documentation

4.4.1.1 void g15r_clearScreen (g15canvas * canvas, int color)

Fills the screen with pixels of color.

Clears the screen and fills it with pixels of color

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>color</i>	Screen will be filled with this color.

Definition at line 80 of file screen.c.

References g15canvas::buffer, and G15_BUFFER_LEN.

```
81 {
82     memset (canvas->buffer, (color ? 0xFF : 0), G15_BUFFER_LEN);
83 }
```

4.4.1.2 int g15r_getPixel (g15canvas * canvas, unsigned int x, unsigned int y)

Gets the value of the pixel at (x, y)

Retrieves the value of the pixel at (x, y)

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x</i>	X offset for pixel to be retrieved.
<i>y</i>	Y offset for pixel to be retrieved.

Definition at line 29 of file screen.c.

References g15canvas::buffer, BYTE_SIZE, G15_LCD_HEIGHT, and G15_LCD_WIDTH.

Referenced by g15r_pixelReverseFill(), and g15r_setPixel().

```
30 {
31     if (x >= G15_LCD_WIDTH || y >= G15_LCD_HEIGHT)
32         return 0;
33
34     unsigned int pixel_offset = y * G15_LCD_WIDTH + x;
35     unsigned int byte_offset = pixel_offset / BYTE_SIZE;
36     unsigned int bit_offset = 7 - (pixel_offset % BYTE_SIZE);
37
38     return (canvas->buffer[byte_offset] & (1 << bit_offset)) >> bit_offset;
39 }
```

4.4.1.3 void g15r_initCanvas (g15canvas * canvas)

Clears the canvas and resets the mode switches.

Clears the screen and resets the mode values for a canvas

Parameters

<i>canvas</i>	A pointer to a g15canvas (p.5) struct
---------------	--

Definition at line 91 of file screen.c.

References `g15canvas::buffer`, `g15canvas::ftLib`, `G15_BUFFER_LEN`, `g15canvas::mode_cache`, `g15canvas::mode_reverse`, and `g15canvas::mode_xor`.

```

92 {
93     memset (canvas->buffer, 0, G15_BUFFER_LEN);
94     canvas->mode_cache = 0;
95     canvas->mode_reverse = 0;
96     canvas->mode_xor = 0;
97 #ifdef TTF_SUPPORT
98     if (FT_Init_FreeType (&canvas->ftLib))
99         printf ("Freetype couldnt initialise\n");
100 #endif
101 }
```

4.4.1.4 void g15r_setPixel (g15canvas * canvas, unsigned int x, unsigned int y, int val)

Sets the value of the pixel at (x, y)

Sets the value of the pixel at (x, y)

Parameters

<i>canvas</i>	A pointer to a g15canvas (p.5) struct in which the buffer to be operated on is found.
<i>x</i>	X offset for pixel to be set.
<i>y</i>	Y offset for pixel to be set.
<i>val</i>	Value to which pixel should be set.

Definition at line 50 of file screen.c.

References `g15canvas::buffer`, `BYTE_SIZE`, `G15_LCD_HEIGHT`, `G15_LCD_WIDTH`, `g15r_getPixel()`, `g15canvas::mode_reverse`, and `g15canvas::mode_xor`.

Referenced by `draw_ttf_char()`, `g15r_drawCircle()`, `g15r_drawIcon()`, `g15r_drawLine()`, `g15r_drawRoundBox()`, `g15r_drawSprite()`, `g15r_pixelBox()`, `g15r_pixelOverlay()`, `g15r_pixelReverseFill()`, `g15r_renderCharacterLarge()`, `g15r_renderCharacterMedium()`, and `g15r_renderCharacterSmall()`.

```

51 {
52     if (x >= G15_LCD_WIDTH || y >= G15_LCD_HEIGHT)
53         return;
54
55     unsigned int pixel_offset = y * G15_LCD_WIDTH + x;
56     unsigned int byte_offset = pixel_offset / BYTE_SIZE;
57     unsigned int bit_offset = 7 - (pixel_offset % BYTE_SIZE);
58
59     if (canvas->mode_xor)
60         val ^= g15r_getPixel (canvas, x, y);
61     if (canvas->mode_reverse)
62         val = !val;
63
64     if (val)
65         canvas->buffer[byte_offset] =
66             canvas->buffer[byte_offset] | 1 << bit_offset;
67     else
68         canvas->buffer[byte_offset] =
69             canvas->buffer[byte_offset] & ~(1 << bit_offset);
70
71 }
```

4.5 text.c File Reference

```
#include "libg15render.h"
```

Functions

- int **calc_ttf_centering** (FT_Face face, char *str)
- int **calc_ttf_right_justify** (FT_Face face, char *str)
- int **calc_ttf_totalstringwidth** (FT_Face face, char *str)
- int **calc_ttf_true_ypos** (FT_Face face, int y, int ttf_fontsize)
- void **draw_ttf_char** (g15canvas *canvas, FT_Bitmap charbitmap, unsigned char character, int x, int y, int color)
- void **draw_ttf_str** (g15canvas *canvas, char *str, int x, int y, int color, FT_Face face)
- void **g15r_renderCharacterLarge** (g15canvas *canvas, int col, int row, unsigned char character, unsigned int sx, unsigned int sy)
Renders a character in the large font at (x, y)
- void **g15r_renderCharacterMedium** (g15canvas *canvas, int col, int row, unsigned char character, unsigned int sx, unsigned int sy)
Renders a character in the meduim font at (x, y)
- void **g15r_renderCharacterSmall** (g15canvas *canvas, int col, int row, unsigned char character, unsigned int sx, unsigned int sy)
Renders a character in the small font at (x, y)
- void **g15r_renderString** (g15canvas *canvas, unsigned char stringOut[], int row, int size, unsigned int sx, unsigned int sy)
Renders a string with font size in row.
- void **g15r_ttfLoad** (g15canvas *canvas, char *fontname, int fontsize, int face_num)
Loads a font through the FreeType2 library.
- void **g15r_ttfPrint** (g15canvas *canvas, int x, int y, int fontsize, int face_num, int color, int center, char *print_string)
Prints a string in a given font.

4.5.1 Function Documentation

4.5.1.1 int calc_ttf_centering (FT_Face face, char * str)

Definition at line 209 of file text.c.

References `calc_ttf_totalstringwidth()`.

Referenced by `g15r_ttfPrint()`.

```

210 {
211     int leftpos;
212
213     leftpos = 80 - (calc_ttf_totalstringwidth (face, str) / 2);
214     if (leftpos < 1)
215         leftpos = 1;
216
217     return leftpos;
218 }
```

4.5.1.2 int calc_ttf_right_justify (FT_Face face, char * str)

Definition at line 221 of file text.c.

References `calc_ttf_totalstringwidth()`.

Referenced by `g15r_ttfPrint()`.

```

222 {
223     int leftpos;
224
225     leftpos = 160 - calc_ttf_totalstringwidth (face, str);
226     if (leftpos < 1)
```



```

227     leftpos = 1;
228
229     return leftpos;
230 }

```

4.5.1.3 int calc_ttf_totalstringwidth (FT_Face *face*, char * *str*)

Definition at line 191 of file text.c.

Referenced by calc_ttf_centering(), and calc_ttf_right_justify().

```

192 {
193     FT_GlyphSlot slot = face->glyph;
194     FT_UInt glyph_index;
195     int i, errcode;
196     unsigned int len = strlen (str);
197     int width = 0;
198
199     for (i = 0; i < len; i++)
200     {
201         glyph_index = FT_Get_Char_Index (face, str[i]);
202         errcode = FT_Load_Glyph (face, glyph_index, 0);
203         width += slot->advance.x >> 6;
204     }
205     return width;
206 }

```

4.5.1.4 int calc_ttf_true_ypos (FT_Face *face*, int *y*, int *ttf_fontsize*)

Definition at line 179 of file text.c.

Referenced by g15r_ttfPrint().

```

180 {
181
182     if (!FT_IS_SCALABLE (face))
183         ttf_fontsize = face->available_sizes->height;
184
185     y += ttf_fontsize * .75;
186
187     return y;
188 }

```

4.5.1.5 void draw_ttf_char (g15canvas* *canvas*, FT_Bitmap *charbitmap*, unsigned char *character*, int *x*, int *y*, int *color*)

Definition at line 233 of file text.c.

References g15canvas::ftLib, and g15r_setPixel().

Referenced by draw_ttf_str().

```

235 {
236     FT_Int char_x, char_y, p, q;
237     FT_Int x_max = x + charbitmap.width;
238     FT_Int y_max = y + charbitmap.rows;
239     static FT_Bitmap tmpbuffer;
240
241     /* convert to 8bit format.. */
242     FT_Bitmap_Convert (canvas->ftLib, &charbitmap, &tmpbuffer, 1);
243
244     for (char_y = y, q = 0; char_y < y_max; char_y++, q++)
245         for (char_x = x, p = 0; char_x < x_max; char_x++, p++)
246             if (tmpbuffer.buffer[q * tmpbuffer.width + p])
247                 g15r_setPixel (canvas, char_x, char_y, color);
248 }

```

4.5.1.6 void draw_ttf_str (g15canvas * canvas, char * str, int x, int y, int color, FT_Face face)

Definition at line 251 of file text.c.

References draw_ttf_char().

Referenced by g15r_ttfPrint().

```

253 {
254     FT_GlyphSlot slot = face->glyph;
255     int i, errcode;
256     unsigned int len = strlen (str);
257
258     for (i = 0; i < len; i++)
259     {
260         errcode =
261             FT_Load_Char (face, str[i],
262                           FT_LOAD_RENDER | FT_LOAD_MONOCHROME |
263                           FT_LOAD_TARGET_MONO);
264         draw_ttf_char (canvas, slot->bitmap, str[i], x + slot->bitmap_left,
265                       y - slot->bitmap_top, color);
266         x += slot->advance.x >> 6;
267     }
268 }
```

4.5.1.7 void g15r_renderCharacterLarge (g15canvas * canvas, int col, int row, unsigned char character, unsigned int sx, unsigned int sy)

Renders a character in the large font at (x, y)

Definition at line 22 of file text.c.

References fontdata_8x8, G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

Referenced by g15r_renderString().

```

25 {
26     int helper = character * 8; /* for our font which is 8x8 */
27
28     int top_left_pixel_x = sx + col * (8); /* 1 pixel spacing */
29     int top_left_pixel_y = sy + row * (8); /* once again 1 pixel spacing */
30
31     int x, y;
32     for (y = 0; y < 8; ++y)
33     {
34         for (x = 0; x < 8; ++x)
35         {
36             char font_entry = fontdata_8x8[helper + y];
37
38             if (font_entry & 1 << (7 - x))
39                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
40                                G15_COLOR_BLACK);
41             else
42                 g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
43                                G15_COLOR_WHITE);
44         }
45     }
46 }
47 }
```

4.5.1.8 void g15r_renderCharacterMedium (g15canvas * canvas, int col, int row, unsigned char character, unsigned int sx, unsigned int sy)

Renders a character in the meduim font at (x, y)

Definition at line 50 of file text.c.

References fontdata_7x5, G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

Referenced by g15r_renderString().

```

53 {
```

```

54  int helper = character * 7 * 5;          /* for our font which is 6x4 */
55
56  int top_left_pixel_x = sx + col * (5);    /* 1 pixel spacing */
57  int top_left_pixel_y = sy + row * (7);    /* once again 1 pixel spacing */
58
59  int x, y;
60  for (y = 0; y < 7; ++y)
61  {
62      for (x = 0; x < 5; ++x)
63      {
64          char font_entry = fontdata_7x5[helper + y * 5 + x];
65          if (font_entry)
66              g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
67                             G15_COLOR_BLACK);
68          else
69              g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
70                             G15_COLOR_WHITE);
71      }
72  }
73  }
74  }

```

4.5.1.9 void g15r_renderCharacterSmall (g15canvas * canvas, int col, int row, unsigned char character, unsigned int sx, unsigned int sy)

Renders a character in the small font at (x, y)

Definition at line 77 of file text.c.

References fontdata_6x4, G15_COLOR_BLACK, G15_COLOR_WHITE, and g15r_setPixel().

Referenced by g15r_renderString().

```

80 {
81  int helper = character * 6 * 4;          /* for our font which is 6x4 */
82
83  int top_left_pixel_x = sx + col * (4);    /* 1 pixel spacing */
84  int top_left_pixel_y = sy + row * (6);    /* once again 1 pixel spacing */
85
86  int x, y;
87  for (y = 0; y < 6; ++y)
88  {
89      for (x = 0; x < 4; ++x)
90      {
91          char font_entry = fontdata_6x4[helper + y * 4 + x];
92          if (font_entry)
93              g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
94                             G15_COLOR_BLACK);
95          else
96              g15r_setPixel (canvas, top_left_pixel_x + x, top_left_pixel_y + y,
97                             G15_COLOR_WHITE);
98      }
99  }
100 }
101 }

```

4.5.1.10 void g15r_renderString (g15canvas * canvas, unsigned char stringOut[], int row, int size, unsigned int sx, unsigned int sy)

Renders a string with font size in row.

Definition at line 104 of file text.c.

References G15_TEXT_LARGE, G15_TEXT_MED, G15_TEXT_SMALL, g15r_renderCharacterLarge(), g15r_renderCharacterMedium(), and g15r_renderCharacterSmall().

```

106 {
107
108  int i = 0;
109  for (i; stringOut[i] != NULL; ++i)
110  {
111      switch (size)
112      {
113          case G15_TEXT_SMALL:

```

```

114         {
115             g15r_renderCharacterSmall (canvas, i, row, stringOut[i], sx, sy);
116             break;
117         }
118         case G15_TEXT_MED:
119         {
120             g15r_renderCharacterMedium (canvas, i, row, stringOut[i], sx, sy);
121             break;
122         }
123         case G15_TEXT_LARGE:
124         {
125             g15r_renderCharacterLarge (canvas, i, row, stringOut[i], sx, sy);
126             break;
127         }
128         default:
129             break;
130     }
131 }
132
133 }

```

4.5.1.11 void g15r_ttfLoad (g15canvas * canvas, char * fontname, int fontsize, int face_num)

Loads a font through the FreeType2 library.

Load a font for use with FreeType2 font support

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>fontname</i>	Absolute pathname to font file to be loaded.
<i>fontsize</i>	Size in points for font to be loaded.
<i>face_num</i>	Slot into which font face will be loaded.

Definition at line 145 of file text.c.

References g15canvas::ftLib, G15_MAX_FACE, g15canvas::ttf_face, and g15canvas::ttf_fontsize.

```

146 {
147     int errcode = 0;
148
149     if (face_num < 0)
150         face_num = 0;
151     if (face_num > G15_MAX_FACE)
152         face_num = G15_MAX_FACE;
153
154     if (canvas->ttf_fontsize[face_num])
155         FT_Done_Face (canvas->ttf_face[face_num][0]);          /* destroy the last face */
156
157     if (!canvas->ttf_fontsize[face_num] && !fontsize)
158         canvas->ttf_fontsize[face_num] = 10;
159     else
160         canvas->ttf_fontsize[face_num] = fontsize;
161
162     errcode =
163         FT_New_Face (canvas->ftLib, fontname, 0, &canvas->ttf_face[face_num][0]);
164     if (errcode)
165     {
166         canvas->ttf_fontsize[face_num] = 0;
167     }
168     else
169     {
170         if (canvas->ttf_fontsize[face_num]
171             && FT_IS_SCALABLE (canvas->ttf_face[face_num][0]))
172             errcode =
173                 FT_Set_Char_Size (canvas->ttf_face[face_num][0], 0,
174                                 canvas->ttf_fontsize[face_num] * 64, 90, 0);
175     }
176 }

```

4.5.1.12 void g15r_ttfPrint (g15canvas * canvas, int x, int y, int fontsize, int face_num, int color, int center, char * print_string)

Prints a string in a given font.

Render a string with a FreeType2 font

Parameters

<i>canvas</i>	A pointer to a g15canvas (p. 5) struct in which the buffer to be operated on is found.
<i>x</i>	initial x position for string.
<i>y</i>	initial y position for string.
<i>fontsize</i>	Size of string in points.
<i>face_num</i>	Font to be used is loaded in this slot.
<i>color</i>	Text will be drawn this color.
<i>center</i>	Text will be centered if center == 1 and right justified if center == 2.
<i>print_string</i>	Pointer to the string to be printed.

Definition at line 283 of file text.c.

References `calc_ttf_centering()`, `calc_ttf_right_justify()`, `calc_ttf_true_ypos()`, `draw_ttf_str()`, `g15canvas::ttf_face`, and `g15canvas::ttf_fontsize`.

```

285 {
286     int errcode = 0;
287
288     if (canvas->ttf_fontsize[face_num])
289     {
290         if (fontsize > 0 && FT_IS_SCALABLE (canvas->ttf_face[face_num][0]))
291         {
292             canvas->ttf_fontsize[face_num] = fontsize;
293             int errcode =
294                 FT_Set_Pixel_Sizes (canvas->ttf_face[face_num][0], 0,
295                                     canvas->ttf_fontsize[face_num]);
296             if (errcode)
297                 printf ("Trouble setting the Glyph size!\n");
298         }
299         y =
300             calc_ttf_true_ypos (canvas->ttf_face[face_num][0], y,
301                                canvas->ttf_fontsize[face_num]);
302         if (center == 1)
303             x = calc_ttf_centering (canvas->ttf_face[face_num][0], print_string);
304         else if (center == 2)
305             x = calc_ttf_right_justify (canvas->ttf_face[face_num][0], print_string);
306         draw_ttf_str (canvas, print_string, x, y, color,
307                      canvas->ttf_face[face_num][0]);
308     }
309 }
```

Index

BYTE_SIZE
 libg15render.h, 11
buffer
 g15canvas, 5

calc_ttf_centering
 text.c, 42
calc_ttf_right_justify
 text.c, 42
calc_ttf_totalstringwidth
 text.c, 43
calc_ttf_true_ypos
 text.c, 43
config.h, 7
 HAVE_DLFCN_H, 7
 HAVE_FT2BUILD_H, 7
 HAVE_INTTYPES_H, 7
 HAVE_LIBG15, 8
 HAVE_LIBM, 8
 HAVE_MEMORY_H, 8
 HAVE_MEMSET, 8
 HAVE_STDINT_H, 8
 HAVE_STDLIB_H, 8
 HAVE_STRING_H, 8
 HAVE_STRINGS_H, 8
 HAVE_SYS_STAT_H, 8
 HAVE_SYS_TYPES_H, 8
 HAVE_UNISTD_H, 8
 PACKAGE, 8
 PACKAGE_BUGREPORT, 9
 PACKAGE_NAME, 9
 PACKAGE_STRING, 9
 PACKAGE_TARNAME, 9
 PACKAGE_VERSION, 9
 STDC_HEADERS, 9
 TTF_SUPPORT, 9
 VERSION, 9

draw_ttf_char
 text.c, 43
draw_ttf_str
 text.c, 43

fontdata_6x4
 libg15render.h, 29
fontdata_7x5
 libg15render.h, 29
fontdata_8x8
 libg15render.h, 29
ftLib
 g15canvas, 5

G15_BUFFER_LEN
 libg15render.h, 11
G15_COLOR_BLACK
 libg15render.h, 11
G15_COLOR_WHITE
 libg15render.h, 11
G15_LCD_HEIGHT
 libg15render.h, 12
G15_LCD_OFFSET
 libg15render.h, 12
G15_LCD_WIDTH
 libg15render.h, 12
G15_MAX_FACE
 libg15render.h, 12
G15_PIXEL_FILL
 libg15render.h, 12
G15_PIXEL_NOFILL
 libg15render.h, 12
G15_TEXT_LARGE
 libg15render.h, 12
G15_TEXT_MED
 libg15render.h, 12
G15_TEXT_SMALL
 libg15render.h, 12
g15canvas, 5
 buffer, 5
 ftLib, 5
 libg15render.h, 13
 mode_cache, 5
 mode_reverse, 6
 mode_xor, 6
 ttf_face, 6
 ttf_fontsize, 6
g15r_clearScreen
 libg15render.h, 13
 screen.c, 40
g15r_drawBar
 libg15render.h, 13
 pixel.c, 30
g15r_drawBigNum
 libg15render.h, 14
 pixel.c, 31
g15r_drawCircle
 libg15render.h, 15
 pixel.c, 32
g15r_drawIcon
 libg15render.h, 16
 pixel.c, 33

- g15r_drawLine
 - libg15render.h, 16
 - pixel.c, 33
- g15r_drawRoundBox
 - libg15render.h, 17
 - pixel.c, 34
- g15r_drawSprite
 - libg15render.h, 18
 - pixel.c, 35
- g15r_getPixel
 - libg15render.h, 19
 - screen.c, 40
- g15r_initCanvas
 - libg15render.h, 19
 - screen.c, 40
- g15r_loadWbmpSplash
 - libg15render.h, 20
 - pixel.c, 36
- g15r_loadWbmpToBuf
 - libg15render.h, 20
 - pixel.c, 36
- g15r_pixelBox
 - libg15render.h, 21
 - pixel.c, 37
- g15r_pixelOverlay
 - libg15render.h, 22
 - pixel.c, 38
- g15r_pixelReverseFill
 - libg15render.h, 22
 - pixel.c, 38
- g15r_renderCharacterLarge
 - libg15render.h, 24
 - text.c, 44
- g15r_renderCharacterMedium
 - libg15render.h, 24
 - text.c, 44
- g15r_renderCharacterSmall
 - libg15render.h, 25
 - text.c, 45
- g15r_renderString
 - libg15render.h, 25
 - text.c, 45
- g15r_setPixel
 - libg15render.h, 26
 - screen.c, 41
- g15r_ttfLoad
 - libg15render.h, 26
 - text.c, 46
- g15r_ttfPrint
 - libg15render.h, 28
 - text.c, 46
- HAVE_DLFCN_H
 - config.h, 7
- HAVE_FT2BUILD_H
 - config.h, 7
- HAVE_INTTYPES_H
 - config.h, 7
- HAVE_LIBG15
 - config.h, 8
- HAVE_LIBM
 - config.h, 8
- HAVE_MEMORY_H
 - config.h, 8
- HAVE_MEMSET
 - config.h, 8
- HAVE_STDINT_H
 - config.h, 8
- HAVE_STDLIB_H
 - config.h, 8
- HAVE_STRING_H
 - config.h, 8
- HAVE_STRINGS_H
 - config.h, 8
- HAVE_SYS_STAT_H
 - config.h, 8
- HAVE_SYS_TYPES_H
 - config.h, 8
- HAVE_UNISTD_H
 - config.h, 8
- libg15render.h, 9
 - BYTE_SIZE, 11
 - fontdata_6x4, 29
 - fontdata_7x5, 29
 - fontdata_8x8, 29
 - G15_BUFFER_LEN, 11
 - G15_COLOR_BLACK, 11
 - G15_COLOR_WHITE, 11
 - G15_LCD_HEIGHT, 12
 - G15_LCD_OFFSET, 12
 - G15_LCD_WIDTH, 12
 - G15_MAX_FACE, 12
 - G15_PIXEL_FILL, 12
 - G15_PIXEL_NOFILL, 12
 - G15_TEXT_LARGE, 12
 - G15_TEXT_MED, 12
 - G15_TEXT_SMALL, 12
 - g15canvas, 13
 - g15r_clearScreen, 13
 - g15r_drawBar, 13
 - g15r_drawBigNum, 14
 - g15r_drawCircle, 15
 - g15r_drawIcon, 16
 - g15r_drawLine, 16
 - g15r_drawRoundBox, 17
 - g15r_drawSprite, 18
 - g15r_getPixel, 19
 - g15r_initCanvas, 19
 - g15r_loadWbmpSplash, 20
 - g15r_loadWbmpToBuf, 20
 - g15r_pixelBox, 21
 - g15r_pixelOverlay, 22
 - g15r_pixelReverseFill, 22
 - g15r_renderCharacterLarge, 24
 - g15r_renderCharacterMedium, 24
 - g15r_renderCharacterSmall, 25
 - g15r_renderString, 25

- g15r_setPixel, 26
- g15r_ttfLoad, 26
- g15r_ttfPrint, 28
- mode_cache
 - g15canvas, 5
- mode_reverse
 - g15canvas, 6
- mode_xor
 - g15canvas, 6
- PACKAGE
 - config.h, 8
- PACKAGE_BUGREPORT
 - config.h, 9
- PACKAGE_NAME
 - config.h, 9
- PACKAGE_STRING
 - config.h, 9
- PACKAGE_TARNAME
 - config.h, 9
- PACKAGE_VERSION
 - config.h, 9
- pixel.c, 29
 - g15r_drawBar, 30
 - g15r_drawBigNum, 31
 - g15r_drawCircle, 32
 - g15r_drawIcon, 33
 - g15r_drawLine, 33
 - g15r_drawRoundBox, 34
 - g15r_drawSprite, 35
 - g15r_loadWbmpSplash, 36
 - g15r_loadWbmpToBuf, 36
 - g15r_pixelBox, 37
 - g15r_pixelOverlay, 38
 - g15r_pixelReverseFill, 38
 - swap, 39
- STDC_HEADERS
 - config.h, 9
- screen.c, 39
 - g15r_clearScreen, 40
 - g15r_getPixel, 40
 - g15r_initCanvas, 40
 - g15r_setPixel, 41
- swap
 - pixel.c, 39
- TTF_SUPPORT
 - config.h, 9
- text.c, 41
 - calc_ttf_centering, 42
 - calc_ttf_right_justify, 42
 - calc_ttf_totalstringwidth, 43
 - calc_ttf_true_ypos, 43
 - draw_ttf_char, 43
 - draw_ttf_str, 43
 - g15r_renderCharacterLarge, 44
 - g15r_renderCharacterMedium, 44
 - g15r_renderCharacterSmall, 45
 - g15r_renderString, 45
 - g15r_ttfLoad, 46
 - g15r_ttfPrint, 46
- ttf_face
 - g15canvas, 6
- ttf_fontsize
 - g15canvas, 6
- VERSION
 - config.h, 9