/\*

u8g\_font.c

U8G Font High Level Interface

Universal 8bit Graphics Library

Copyright (c) 2011, olikraus@gmail.com

All rights reserved.

Redistribution and use in source and binary forms, with or without modification,

are permitted provided that the following conditions are met:

\* Redistributions of source code must retain the above copyright notice, this list

of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright notice, this

list of conditions and the following disclaimer in the documentation and/or other

materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND

CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,

INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT

NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,

STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF

ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

#include "u8g.h"

/\* font api \*/

/\* pointer to the start adress of the glyph, points to progmem area \*/

typedef void \* u8g\_glyph\_t;

/\* size of the font data structure, there is no struct or class... \*/

#define U8G\_FONT\_DATA\_STRUCT\_SIZE 17

/\*

... instead the fields of the font data structure are accessed directly by offset

font information

offset

0 font format

1 FONTBOUNDINGBOX width unsigned

2 FONTBOUNDINGBOX height unsigned

3 FONTBOUNDINGBOX x-offset signed

4 FONTBOUNDINGBOX y-offset signed

5 capital A height unsigned

6 start 'A'

8 start 'a'

10 encoding start

11 encoding end

12 descent 'g' negative: below baseline

13 font max ascent

14 font min decent negative: below baseline

15 font xascent

16 font xdecent negative: below baseline

\*/

/\* use case: What is the width and the height of the minimal box into which string s fints? \*/

void u8g\_font\_GetStrSize(const void \*font, const char \*s, u8g\_uint\_t \*width, u8g\_uint\_t \*height);

void u8g\_font\_GetStrSizeP(const void \*font, const char \*s, u8g\_uint\_t \*width, u8g\_uint\_t \*height);

/\* use case: lower left edge of a minimal box is known, what is the correct x, y position for the string draw procedure \*/

void u8g\_font\_AdjustXYToDraw(const void \*font, const char \*s, u8g\_uint\_t \*x, u8g\_uint\_t \*y);

void u8g\_font\_AdjustXYToDrawP(const void \*font, const char \*s, u8g\_uint\_t \*x, u8g\_uint\_t \*y);

/\* use case: Baseline origin known, return minimal box \*/

void u8g\_font\_GetStrMinBox(u8g\_t \*u8g, const void \*font, const char \*s, u8g\_uint\_t \*x, u8g\_uint\_t \*y, u8g\_uint\_t \*width, u8g\_uint\_t \*height);

/\* procedures \*/

/\*========================================================================\*/

/\* low level byte and word access \*/

/\* removed NOINLINE, because it leads to smaller code, might also be faster \*/

//static uint8\_t u8g\_font\_get\_byte(const u8g\_fntpgm\_uint8\_t \*font, uint8\_t offset) U8G\_NOINLINE;

static uint8\_t u8g\_font\_get\_byte(const u8g\_fntpgm\_uint8\_t \*font, uint8\_t offset)

{

font += offset;

return u8g\_pgm\_read( (u8g\_pgm\_uint8\_t \*)font );

}

static uint16\_t u8g\_font\_get\_word(const u8g\_fntpgm\_uint8\_t \*font, uint8\_t offset) U8G\_NOINLINE;

static uint16\_t u8g\_font\_get\_word(const u8g\_fntpgm\_uint8\_t \*font, uint8\_t offset)

{

uint16\_t pos;

font += offset;

pos = u8g\_pgm\_read( (u8g\_pgm\_uint8\_t \*)font );

font++;

pos <<= 8;

pos += u8g\_pgm\_read( (u8g\_pgm\_uint8\_t \*)font);

return pos;

}

/\*========================================================================\*/

/\* direct access on the font \*/

static uint8\_t u8g\_font\_GetFormat(const u8g\_fntpgm\_uint8\_t \*font) U8G\_NOINLINE;

static uint8\_t u8g\_font\_GetFormat(const u8g\_fntpgm\_uint8\_t \*font)

{

return u8g\_font\_get\_byte(font, 0);

}

static uint8\_t u8g\_font\_GetFontGlyphStructureSize(const u8g\_fntpgm\_uint8\_t \*font) U8G\_NOINLINE;

static uint8\_t u8g\_font\_GetFontGlyphStructureSize(const u8g\_fntpgm\_uint8\_t \*font)

{

switch(u8g\_font\_GetFormat(font))

{

case 0: return 6;

case 1: return 3;

case 2: return 6;

}

return 3;

}

static uint8\_t u8g\_font\_GetBBXWidth(const void \*font)

{

return u8g\_font\_get\_byte(font, 1);

}

static uint8\_t u8g\_font\_GetBBXHeight(const void \*font)

{

return u8g\_font\_get\_byte(font, 2);

}

static int8\_t u8g\_font\_GetBBXOffX(const void \*font)

{

return u8g\_font\_get\_byte(font, 3);

}

static int8\_t u8g\_font\_GetBBXOffY(const void \*font)

{

return u8g\_font\_get\_byte(font, 4);

}

uint8\_t u8g\_font\_GetCapitalAHeight(const void \*font)

{

return u8g\_font\_get\_byte(font, 5);

}

uint16\_t u8g\_font\_GetEncoding65Pos(const void \*font) U8G\_NOINLINE;

uint16\_t u8g\_font\_GetEncoding65Pos(const void \*font)

{

return u8g\_font\_get\_word(font, 6);

}

uint16\_t u8g\_font\_GetEncoding97Pos(const void \*font) U8G\_NOINLINE;

uint16\_t u8g\_font\_GetEncoding97Pos(const void \*font)

{

return u8g\_font\_get\_word(font, 8);

}

uint8\_t u8g\_font\_GetFontStartEncoding(const void \*font)

{

return u8g\_font\_get\_byte(font, 10);

}

uint8\_t u8g\_font\_GetFontEndEncoding(const void \*font)

{

return u8g\_font\_get\_byte(font, 11);

}

int8\_t u8g\_font\_GetLowerGDescent(const void \*font)

{

return u8g\_font\_get\_byte(font, 12);

}

int8\_t u8g\_font\_GetFontAscent(const void \*font)

{

return u8g\_font\_get\_byte(font, 13);

}

int8\_t u8g\_font\_GetFontDescent(const void \*font)

{

return u8g\_font\_get\_byte(font, 14);

}

int8\_t u8g\_font\_GetFontXAscent(const void \*font)

{

return u8g\_font\_get\_byte(font, 15);

}

int8\_t u8g\_font\_GetFontXDescent(const void \*font)

{

return u8g\_font\_get\_byte(font, 16);

}

/\* return the data start for a font and the glyph pointer \*/

static uint8\_t \*u8g\_font\_GetGlyphDataStart(const void \*font, u8g\_glyph\_t g)

{

return ((u8g\_fntpgm\_uint8\_t \*)g) + u8g\_font\_GetFontGlyphStructureSize(font);

}

/\* calculate the overall length of the font, only used to create the picture for the google wiki \*/

size\_t u8g\_font\_GetSize(const void \*font)

{

uint8\_t \*p = (uint8\_t \*)(font);

uint8\_t font\_format = u8g\_font\_GetFormat(font);

uint8\_t data\_structure\_size = u8g\_font\_GetFontGlyphStructureSize(font);

uint8\_t start, end;

uint8\_t i;

uint8\_t mask = 255;

start = u8g\_font\_GetFontStartEncoding(font);

end = u8g\_font\_GetFontEndEncoding(font);

if ( font\_format == 1 )

mask = 15;

p += U8G\_FONT\_DATA\_STRUCT\_SIZE; /\* skip font general information \*/

i = start;

for(;;)

{

if ( u8g\_pgm\_read((u8g\_pgm\_uint8\_t \*)(p)) == 255 )

{

p += 1;

}

else

{

p += u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)(p)) + 2 ) & mask;

p += data\_structure\_size;

}

if ( i == end )

break;

i++;

}

return p - (uint8\_t \*)font;

}

/\*========================================================================\*/

/\* u8g interface, font access \*/

uint8\_t u8g\_GetFontBBXWidth(u8g\_t \*u8g)

{

return u8g\_font\_GetBBXWidth(u8g->font);

}

uint8\_t u8g\_GetFontBBXHeight(u8g\_t \*u8g)

{

return u8g\_font\_GetBBXHeight(u8g->font);

}

int8\_t u8g\_GetFontBBXOffX(u8g\_t \*u8g) U8G\_NOINLINE;

int8\_t u8g\_GetFontBBXOffX(u8g\_t \*u8g)

{

return u8g\_font\_GetBBXOffX(u8g->font);

}

int8\_t u8g\_GetFontBBXOffY(u8g\_t \*u8g) U8G\_NOINLINE;

int8\_t u8g\_GetFontBBXOffY(u8g\_t \*u8g)

{

return u8g\_font\_GetBBXOffY(u8g->font);

}

uint8\_t u8g\_GetFontCapitalAHeight(u8g\_t \*u8g) U8G\_NOINLINE;

uint8\_t u8g\_GetFontCapitalAHeight(u8g\_t \*u8g)

{

return u8g\_font\_GetCapitalAHeight(u8g->font);

}

/\*========================================================================\*/

/\* glyph handling \*/

static void u8g\_CopyGlyphDataToCache(u8g\_t \*u8g, u8g\_glyph\_t g)

{

uint8\_t tmp;

switch( u8g\_font\_GetFormat(u8g->font) )

{

case 0:

case 2:

/\*

format 0

glyph information

offset

0 BBX width unsigned

1 BBX height unsigned

2 data size unsigned (BBX width + 7)/8 \* BBX height

3 DWIDTH signed

4 BBX xoffset signed

5 BBX yoffset signed

byte 0 == 255 indicates empty glyph

\*/

u8g->glyph\_width = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 0 );

u8g->glyph\_height = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 1 );

u8g->glyph\_dx = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 3 );

u8g->glyph\_x = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 4 );

u8g->glyph\_y = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 5 );

break;

case 1:

default:

/\*

format 1

0 BBX xoffset signed --> upper 4 Bit

0 BBX yoffset signed --> lower 4 Bit

1 BBX width unsigned --> upper 4 Bit

1 BBX height unsigned --> lower 4 Bit

2 data size unsigned -(BBX width + 7)/8 \* BBX height --> lower 4 Bit

2 DWIDTH signed --> upper 4 Bit

byte 0 == 255 indicates empty glyph

\*/

tmp = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 0 );

u8g->glyph\_y = tmp & 15;

u8g->glyph\_y-=2;

tmp >>= 4;

u8g->glyph\_x = tmp;

tmp = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 1 );

u8g->glyph\_height = tmp & 15;

tmp >>= 4;

u8g->glyph\_width = tmp;

tmp = u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)g) + 2 );

tmp >>= 4;

u8g->glyph\_dx = tmp;

break;

}

}

//void u8g\_FillEmptyGlyphCache(u8g\_t \*u8g) U8G\_NOINLINE;

static void u8g\_FillEmptyGlyphCache(u8g\_t \*u8g)

{

u8g->glyph\_dx = 0;

u8g->glyph\_width = 0;

u8g->glyph\_height = 0;

u8g->glyph\_x = 0;

u8g->glyph\_y = 0;

}

/\*

Find (with some speed optimization) and return a pointer to the glyph data structure

Also uncompress (format 1) and copy the content of the data structure to the u8g structure

\*/

u8g\_glyph\_t u8g\_GetGlyph(u8g\_t \*u8g, uint8\_t requested\_encoding)

{

uint8\_t \*p = (uint8\_t \*)(u8g->font);

uint8\_t font\_format = u8g\_font\_GetFormat(u8g->font);

uint8\_t data\_structure\_size = u8g\_font\_GetFontGlyphStructureSize(u8g->font);

uint8\_t start, end;

uint16\_t pos;

uint8\_t i;

uint8\_t mask = 255;

if ( font\_format == 1 )

mask = 15;

start = u8g\_font\_GetFontStartEncoding(u8g->font);

end = u8g\_font\_GetFontEndEncoding(u8g->font);

pos = u8g\_font\_GetEncoding97Pos(u8g->font);

if ( requested\_encoding >= 97 && pos > 0 )

{

p+= pos;

start = 97;

}

else

{

pos = u8g\_font\_GetEncoding65Pos(u8g->font);

if ( requested\_encoding >= 65 && pos > 0 )

{

p+= pos;

start = 65;

}

else

p += U8G\_FONT\_DATA\_STRUCT\_SIZE; /\* skip font general information \*/

}

if ( requested\_encoding > end )

{

u8g\_FillEmptyGlyphCache(u8g);

return NULL; /\* not found \*/

}

i = start;

if ( i <= end )

{

for(;;)

{

if ( u8g\_pgm\_read((u8g\_pgm\_uint8\_t \*)(p)) == 255 )

{

p += 1;

}

else

{

if ( i == requested\_encoding )

{

u8g\_CopyGlyphDataToCache(u8g, p);

return p;

}

p += u8g\_pgm\_read( ((u8g\_pgm\_uint8\_t \*)(p)) + 2 ) & mask;

p += data\_structure\_size;

}

if ( i == end )

break;

i++;

}

}

u8g\_FillEmptyGlyphCache(u8g);

return NULL;

}

uint8\_t u8g\_IsGlyph(u8g\_t \*u8g, uint8\_t requested\_encoding)

{

if ( u8g\_GetGlyph(u8g, requested\_encoding) != NULL )

return 1;

return 0;

}

int8\_t u8g\_GetGlyphDeltaX(u8g\_t \*u8g, uint8\_t requested\_encoding)

{

if ( u8g\_GetGlyph(u8g, requested\_encoding) == NULL )

return 0; /\* should never happen, so return something \*/

return u8g->glyph\_dx;

}

/\*========================================================================\*/

/\* glyph drawing procedures \*/

#ifdef OBSOLETE

/\*

Draw a glyph

x,y: left baseline position of the glyph

\*/

int8\_t u8g\_DrawGlyphDir(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, uint8\_t encoding)

{

u8g\_glyph\_t g;

uint8\_t w, h, i, j;

const u8g\_pgm\_uint8\_t \*data;

uint8\_t bytes\_per\_line;

u8g\_uint\_t ix, iy;

g = u8g\_GetGlyph(u8g, encoding);

if ( g == NULL )

return 0;

w = u8g->glyph\_width;

h = u8g->glyph\_height;

bytes\_per\_line = w;

bytes\_per\_line += 7;

bytes\_per\_line /= 8;

data = u8g\_font\_GetGlyphDataStart(u8g->font, g);

switch(dir)

{

case 0:

x += u8g->glyph\_x;

y -= u8g->glyph\_y;

y--;

//u8g\_DrawFrame(u8g, x, y-h+1, w, h);

if ( u8g\_IsBBXIntersection(u8g, x, y-h+1, w, h) == 0 )

return u8g->glyph\_dx;

iy = y;

iy -= h;

iy++;

for( j = 0; j < h; j++ )

{

ix = x;

for( i = 0; i < bytes\_per\_line; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, dir, u8g\_pgm\_read(data));

data++;

ix+=8;

}

iy++;

}

break;

case 1:

x += u8g->glyph\_y;

x++;

y += u8g->glyph\_x;

//printf("enc %d, dir %d, x %d, y %d, w %d, h %d\n", encoding, dir, x, y, w, h);

//u8g\_DrawFrame(u8g, x, y, h, w);

if ( u8g\_IsBBXIntersection(u8g, x, y, h, w) == 0 )

return u8g->glyph\_dx;

ix = x;

ix += h;

ix--;

for( j = 0; j < h; j++ )

{

iy = y;

for( i = 0; i < bytes\_per\_line; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, dir, u8g\_pgm\_read(data));

data++;

iy+=8;

}

ix--;

}

break;

case 2:

x -= u8g->glyph\_x;

y += u8g->glyph\_y;

y++;

if ( u8g\_IsBBXIntersection(u8g, x-w-1, y, w, h) == 0 )

return u8g->glyph\_dx;

iy = y;

iy += h;

iy--;

for( j = 0; j < h; j++ )

{

ix = x;

for( i = 0; i < bytes\_per\_line; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, dir, u8g\_pgm\_read(data));

data++;

ix-=8;

}

iy--;

}

break;

case 3:

x -= u8g->glyph\_y;

x--;

y -= u8g->glyph\_x;

if ( u8g\_IsBBXIntersection(u8g, x-h-1, y-w-1, h, w) == 0 )

return u8g->glyph\_dx;

ix = x;

ix -= h;

ix++;

for( j = 0; j < h; j++ )

{

iy = y;

for( i = 0; i < bytes\_per\_line; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, dir, u8g\_pgm\_read(data));

data++;

iy-=8;

}

ix++;

}

break;

}

return u8g->glyph\_dx;

}

#endif

int8\_t u8g\_draw\_glyph(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

const u8g\_pgm\_uint8\_t \*data;

uint8\_t w, h;

uint8\_t i, j;

u8g\_uint\_t ix, iy;

{

u8g\_glyph\_t g = u8g\_GetGlyph(u8g, encoding);

if ( g == NULL )

return 0;

data = u8g\_font\_GetGlyphDataStart(u8g->font, g);

}

w = u8g->glyph\_width;

h = u8g->glyph\_height;

x += u8g->glyph\_x;

y -= u8g->glyph\_y;

y--;

if ( u8g\_IsBBXIntersection(u8g, x, y-h+1, w, h) == 0 )

return u8g->glyph\_dx;

/\* now, w is reused as bytes per line \*/

w += 7;

w /= 8;

iy = y;

iy -= h;

iy++;

for( j = 0; j < h; j++ )

{

ix = x;

for( i = 0; i < w; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, 0, u8g\_pgm\_read(data));

data++;

ix+=8;

}

iy++;

}

return u8g->glyph\_dx;

}

int8\_t u8g\_DrawGlyph(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

y += u8g->font\_calc\_vref(u8g);

return u8g\_draw\_glyph(u8g, x, y, encoding);

}

int8\_t u8g\_draw\_glyph90(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

const u8g\_pgm\_uint8\_t \*data;

uint8\_t w, h;

uint8\_t i, j;

u8g\_uint\_t ix, iy;

{

u8g\_glyph\_t g = u8g\_GetGlyph(u8g, encoding);

if ( g == NULL )

return 0;

data = u8g\_font\_GetGlyphDataStart(u8g->font, g);

}

w = u8g->glyph\_width;

h = u8g->glyph\_height;

x += u8g->glyph\_y;

x++;

y += u8g->glyph\_x;

if ( u8g\_IsBBXIntersection(u8g, x, y, h, w) == 0 )

return u8g->glyph\_dx;

/\* now, w is reused as bytes per line \*/

w += 7;

w /= 8;

ix = x;

ix += h;

ix--;

for( j = 0; j < h; j++ )

{

iy = y;

for( i = 0; i < w; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, 1, u8g\_pgm\_read(data));

data++;

iy+=8;

}

ix--;

}

return u8g->glyph\_dx;

}

int8\_t u8g\_DrawGlyph90(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

x -= u8g->font\_calc\_vref(u8g);

return u8g\_draw\_glyph90(u8g, x, y, encoding);

}

int8\_t u8g\_draw\_glyph180(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

const u8g\_pgm\_uint8\_t \*data;

uint8\_t w, h;

uint8\_t i, j;

u8g\_uint\_t ix, iy;

{

u8g\_glyph\_t g = u8g\_GetGlyph(u8g, encoding);

if ( g == NULL )

return 0;

data = u8g\_font\_GetGlyphDataStart(u8g->font, g);

}

w = u8g->glyph\_width;

h = u8g->glyph\_height;

x -= u8g->glyph\_x;

y += u8g->glyph\_y;

y++;

if ( u8g\_IsBBXIntersection(u8g, x-(w-1), y, w, h) == 0 )

return u8g->glyph\_dx;

/\* now, w is reused as bytes per line \*/

w += 7;

w /= 8;

iy = y;

iy += h;

iy--;

for( j = 0; j < h; j++ )

{

ix = x;

for( i = 0; i < w; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, 2, u8g\_pgm\_read(data));

data++;

ix-=8;

}

iy--;

}

return u8g->glyph\_dx;

}

int8\_t u8g\_DrawGlyph180(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

y -= u8g->font\_calc\_vref(u8g);

return u8g\_draw\_glyph180(u8g, x, y, encoding);

}

int8\_t u8g\_draw\_glyph270(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

const u8g\_pgm\_uint8\_t \*data;

uint8\_t w, h;

uint8\_t i, j;

u8g\_uint\_t ix, iy;

{

u8g\_glyph\_t g = u8g\_GetGlyph(u8g, encoding);

if ( g == NULL )

return 0;

data = u8g\_font\_GetGlyphDataStart(u8g->font, g);

}

w = u8g->glyph\_width;

h = u8g->glyph\_height;

x -= u8g->glyph\_y;

x--;

y -= u8g->glyph\_x;

if ( u8g\_IsBBXIntersection(u8g, x-(h-1), y-(w-1), h, w) == 0 )

return u8g->glyph\_dx;

/\* now, w is reused as bytes per line \*/

w += 7;

w /= 8;

ix = x;

ix -= h;

ix++;

for( j = 0; j < h; j++ )

{

iy = y;

for( i = 0; i < w; i++ )

{

u8g\_Draw8Pixel(u8g, ix, iy, 3, u8g\_pgm\_read(data));

data++;

iy-=8;

}

ix++;

}

return u8g->glyph\_dx;

}

int8\_t u8g\_DrawGlyph270(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

x += u8g->font\_calc\_vref(u8g);

return u8g\_draw\_glyph270(u8g, x, y, encoding);

}

#ifdef OBSOLETE

/\*

Draw a glyph

x,y: lower left corner of the font bounding box

\*/

int8\_t u8g\_DrawGlyphFontBBX(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, uint8\_t encoding)

{

/\* TODO: apply "dir" \*/

x -= u8g\_GetFontBBXOffX(u8g);

y += u8g\_GetFontBBXOffY(u8g);

return u8g\_DrawGlyphDir(u8g, x, y, dir, encoding);

}

#endif

/\*========================================================================\*/

/\* string drawing procedures \*/

u8g\_uint\_t u8g\_DrawStr(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const char \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

//u8g\_uint\_t u8g\_GetStrWidth(u8g, s);

//u8g\_font\_GetFontAscent(u8g->font)-u8g\_font\_GetFontDescent(u8g->font);

y += u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_draw\_glyph(u8g, x, y, \*s);

x += d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStr90(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const char \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

x -= u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_draw\_glyph90(u8g, x, y, \*s);

y += d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStr180(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const char \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

y -= u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_draw\_glyph180(u8g, x, y, \*s);

x -= d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStr270(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const char \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

x += u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_draw\_glyph270(u8g, x, y, \*s);

y -= d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStrDir(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, const char \*s)

{

switch(dir)

{

case 0:

return u8g\_DrawStr(u8g, x, y, s);

case 1:

return u8g\_DrawStr90(u8g, x, y, s);

case 2:

return u8g\_DrawStr180(u8g, x, y, s);

case 3:

return u8g\_DrawStr270(u8g, x, y, s);

}

return 0;

}

u8g\_uint\_t u8g\_DrawStrP(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const u8g\_pgm\_uint8\_t \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

uint8\_t c;

y += u8g->font\_calc\_vref(u8g);

for(;;)

{

c = u8g\_pgm\_read(s);

if ( c == '\0' )

break;

d = u8g\_draw\_glyph(u8g, x, y, c);

x += d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStr90P(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const u8g\_pgm\_uint8\_t \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

x -= u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_DrawGlyph90(u8g, x, y, u8g\_pgm\_read(s));

y += d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStr180P(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const u8g\_pgm\_uint8\_t \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

y -= u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_DrawGlyph180(u8g, x, y, u8g\_pgm\_read(s));

x -= d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStr270P(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const u8g\_pgm\_uint8\_t \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

x += u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_DrawGlyph270(u8g, x, y, u8g\_pgm\_read(s));

y -= d;

t += d;

s++;

}

return t;

}

u8g\_uint\_t u8g\_DrawStrFontBBX(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, const char \*s)

{

x -= u8g\_GetFontBBXOffX(u8g);

y += u8g\_GetFontBBXOffY(u8g);

return u8g\_DrawStrDir(u8g, x, y, dir, s);

}

/\* still used by picgen.c, dir argument is ignored \*/

int8\_t u8g\_DrawGlyphFontBBX(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, uint8\_t encoding)

{

x -= u8g\_GetFontBBXOffX(u8g);

y += u8g\_GetFontBBXOffY(u8g);

u8g\_draw\_glyph(u8g, x, y, encoding);

return 0;

}

/\*========================================================================\*/

/\* set ascent/descent for reference point calculation \*/

void u8g\_UpdateRefHeight(u8g\_t \*u8g)

{

uint16\_t ls;

if ( u8g->font == NULL )

return;

if ( u8g->font\_height\_mode == U8G\_FONT\_HEIGHT\_MODE\_TEXT )

{

u8g->font\_ref\_ascent = u8g\_font\_GetCapitalAHeight(u8g->font);

u8g->font\_ref\_descent = u8g\_font\_GetLowerGDescent(u8g->font);

}

else if ( u8g->font\_height\_mode == U8G\_FONT\_HEIGHT\_MODE\_XTEXT )

{

u8g->font\_ref\_ascent = u8g\_font\_GetFontXAscent(u8g->font);

u8g->font\_ref\_descent = u8g\_font\_GetFontXDescent(u8g->font);

}

else

{

u8g->font\_ref\_ascent = u8g\_font\_GetFontAscent(u8g->font);

u8g->font\_ref\_descent = u8g\_font\_GetFontDescent(u8g->font);

}

ls = u8g->font\_ref\_ascent - u8g->font\_ref\_descent;

if ( u8g->font\_line\_spacing\_factor != 64 )

{

ls &= 255;

ls \*= u8g->font\_line\_spacing\_factor;

ls >>= 6;

}

u8g->line\_spacing = ls;

}

void u8g\_SetFontRefHeightText(u8g\_t \*u8g)

{

u8g->font\_height\_mode = U8G\_FONT\_HEIGHT\_MODE\_TEXT;

u8g\_UpdateRefHeight(u8g);

}

void u8g\_SetFontRefHeightExtendedText(u8g\_t \*u8g)

{

u8g->font\_height\_mode = U8G\_FONT\_HEIGHT\_MODE\_XTEXT;

u8g\_UpdateRefHeight(u8g);

}

void u8g\_SetFontRefHeightAll(u8g\_t \*u8g)

{

u8g->font\_height\_mode = U8G\_FONT\_HEIGHT\_MODE\_ALL;

u8g\_UpdateRefHeight(u8g);

}

/\* factor = 64: linespaceing == ascent and descent \*/

void u8g\_SetFontLineSpacingFactor(u8g\_t \*u8g, uint8\_t factor)

{

u8g->font\_line\_spacing\_factor = factor;

u8g\_UpdateRefHeight(u8g);

}

/\*========================================================================\*/

/\* callback procedures to correct the y position \*/

u8g\_uint\_t u8g\_font\_calc\_vref\_font(u8g\_t \*u8g)

{

return 0;

}

void u8g\_SetFontPosBaseline(u8g\_t \*u8g)

{

u8g->font\_calc\_vref = u8g\_font\_calc\_vref\_font;

}

u8g\_uint\_t u8g\_font\_calc\_vref\_bottom(u8g\_t \*u8g)

{

/\* y += (u8g\_uint\_t)(u8g\_int\_t)(u8g->font\_ref\_descent); \*/

return (u8g\_uint\_t)(u8g\_int\_t)(u8g->font\_ref\_descent);

}

void u8g\_SetFontPosBottom(u8g\_t \*u8g)

{

u8g->font\_calc\_vref = u8g\_font\_calc\_vref\_bottom;

}

u8g\_uint\_t u8g\_font\_calc\_vref\_top(u8g\_t \*u8g)

{

u8g\_uint\_t tmp;

/\* reference pos is one pixel above the upper edge of the reference glyph \*/

/\*

y += (u8g\_uint\_t)(u8g\_int\_t)(u8g->font\_ref\_ascent);

y++;

\*/

tmp = (u8g\_uint\_t)(u8g\_int\_t)(u8g->font\_ref\_ascent);

tmp++;

return tmp;

}

void u8g\_SetFontPosTop(u8g\_t \*u8g)

{

u8g->font\_calc\_vref = u8g\_font\_calc\_vref\_top;

}

u8g\_uint\_t u8g\_font\_calc\_vref\_center(u8g\_t \*u8g)

{

int8\_t tmp;

tmp = u8g->font\_ref\_ascent;

tmp -= u8g->font\_ref\_descent;

tmp /= 2;

tmp += u8g->font\_ref\_descent;

/\* y += (u8g\_uint\_t)(u8g\_int\_t)(tmp); \*/

return tmp;

}

void u8g\_SetFontPosCenter(u8g\_t \*u8g)

{

u8g->font\_calc\_vref = u8g\_font\_calc\_vref\_center;

}

/\*========================================================================\*/

/\* string pixel width calculation \*/

char u8g\_font\_get\_char(const void \*s)

{

return \*(const char \*)(s);

}

char u8g\_font\_get\_charP(const void \*s)

{

return u8g\_pgm\_read(s);

}

typedef char (\*u8g\_font\_get\_char\_fn)(const void \*s);

u8g\_uint\_t u8g\_font\_calc\_str\_pixel\_width(u8g\_t \*u8g, const char \*s, u8g\_font\_get\_char\_fn get\_char )

{

u8g\_uint\_t w;

uint8\_t enc;

/\* reset the total minimal width to zero, this will be expanded during calculation \*/

w = 0;

enc = get\_char(s);

/\* check for empty string, width is already 0 \*/

if ( enc == '\0' )

{

return w;

}

/\* get the glyph information of the first char. This must be valid, because we already checked for the empty string \*/

/\* if \*s is not inside the font, then the cached parameters of the glyph are all zero \*/

u8g\_GetGlyph(u8g, enc);

/\* strlen(s) == 1: width = width(s[0]) \*/

/\* strlen(s) == 2: width = - offx(s[0]) + deltax(s[0]) + offx(s[1]) + width(s[1]) \*/

/\* strlen(s) == 3: width = - offx(s[0]) + deltax(s[0]) + deltax(s[1]) + offx(s[2]) + width(s[2]) \*/

/\* assume that the string has size 2 or more, than start with negative offset-x \*/

/\* for string with size 1, this will be nullified after the loop \*/

w = -u8g->glyph\_x;

for(;;)

{

/\* check and stop if the end of the string is reached \*/

s++;

if ( get\_char(s) == '\0' )

break;

/\* if there are still more characters, add the delta to the next glyph \*/

w += u8g->glyph\_dx;

/\* store the encoding in a local variable, used also after the for(;;) loop \*/

enc = get\_char(s);

/\* load the next glyph information \*/

u8g\_GetGlyph(u8g, enc);

}

/\* finally calculate the width of the last char \*/

/\* here is another exception, if the last char is a black, use the dx value instead \*/

if ( enc != ' ' )

{

/\* if g was not updated in the for loop (strlen() == 1), then the initial offset x gets removed \*/

w += u8g->glyph\_width;

w += u8g->glyph\_x;

}

else

{

w += u8g->glyph\_dx;

}

return w;

}

u8g\_uint\_t u8g\_GetStrPixelWidth(u8g\_t \*u8g, const char \*s)

{

return u8g\_font\_calc\_str\_pixel\_width(u8g, s, u8g\_font\_get\_char);

}

u8g\_uint\_t u8g\_GetStrPixelWidthP(u8g\_t \*u8g, const u8g\_pgm\_uint8\_t \*s)

{

return u8g\_font\_calc\_str\_pixel\_width(u8g, (const char \*)s, u8g\_font\_get\_charP);

}

int8\_t u8g\_GetStrX(u8g\_t \*u8g, const char \*s)

{

u8g\_GetGlyph(u8g, \*s);

return u8g->glyph\_x;

}

int8\_t u8g\_GetStrXP(u8g\_t \*u8g, const u8g\_pgm\_uint8\_t \*s)

{

u8g\_GetGlyph(u8g, u8g\_pgm\_read(s));

return u8g->glyph\_x;

}

/\*========================================================================\*/

/\* string width calculation \*/

u8g\_uint\_t u8g\_GetStrWidth(u8g\_t \*u8g, const char \*s)

{

u8g\_uint\_t w;

uint8\_t encoding;

/\* reset the total width to zero, this will be expanded during calculation \*/

w = 0;

for(;;)

{

encoding = \*s;

if ( encoding == 0 )

break;

/\* load glyph information \*/

u8g\_GetGlyph(u8g, encoding);

w += u8g->glyph\_dx;

/\* goto next char \*/

s++;

}

return w;

}

u8g\_uint\_t u8g\_GetStrWidthP(u8g\_t \*u8g, const u8g\_pgm\_uint8\_t \*s)

{

u8g\_uint\_t w;

uint8\_t encoding;

/\* reset the total width to zero, this will be expanded during calculation \*/

w = 0;

for(;;)

{

encoding = u8g\_pgm\_read(s);

if ( encoding == 0 )

break;

/\* load glyph information \*/

u8g\_GetGlyph(u8g, encoding);

w += u8g->glyph\_dx;

/\* goto next char \*/

s++;

}

return w;

}

/\*========================================================================\*/

/\* calculation of font/glyph/string characteristics \*/

/\*

Description:

Calculate parameter for the minimal bounding box on a given string

Output

buf->y\_min extend of the lower left edge if the string below (y\_min<0) or above (y\_min>0) baseline (descent)

buf->y\_max extend of the upper left edge if the string below (y\_min<0) or above (y\_min>0) baseline (ascent)

buf->w the width of the string

\*/

struct u8g\_str\_size\_struct

{

int8\_t y\_min; /\* descent \*/

int8\_t y\_max; /\* ascent \*/

int8\_t x, y; /\* the reference point of the font (negated!) \*/

u8g\_uint\_t w; /\* width of the overall string \*/

};

typedef struct u8g\_str\_size\_struct u8g\_str\_size\_t;

static void u8g\_font\_calc\_str\_min\_box(u8g\_t \*u8g, const char \*s, u8g\_str\_size\_t \*buf)

{

/\* u8g\_glyph\_t g; \*/

int8\_t tmp;

/\* reset the total minimal width to zero, this will be expanded during calculation \*/

buf->w = 0;

/\* check for empty string, width is already 0, but also reset y\_min and y\_max to 0 \*/

if ( \*s == '\0' )

{

buf->y\_min = 0;

buf->y\_max = 0;

buf->x = 0;

buf->y = 0;

return;

}

/\* reset y\_min to the largest possible value. Later we search for the smallest value \*/

/\* y\_min contains the position [pixel] of the lower left edge of the glyph above (y\_min>0) or below (y\_min<0) baseline \*/

buf->y\_min = 127;

/\* reset y\_max to the smallest possible value. Later we search for the highest value \*/

/\* y\_max contains the position [pixel] of the upper left edge of the glyph above (y\_max>0) or below (y\_max<0) baseline \*/

buf->y\_max = -128;

/\* get the glyph information of the first char. This must be valid, because we already checked for the empty string \*/

u8g\_GetGlyph(u8g, \*s);

/\* strlen(s) == 1: width = width(s[0]) \*/

/\* strlen(s) == 2: width = - offx(s[0]) + deltax(s[0]) + offx(s[1]) + width(s[1]) \*/

/\* strlen(s) == 3: width = - offx(s[0]) + deltax(s[0]) + deltax(s[1]) + offx(s[2]) + width(s[2]) \*/

/\* assume that the string has size 2 or more, than start with negative offset-x \*/

/\* for string with size 1, this will be nullified after the loop \*/

// buf->w = - u8g\_font\_GetGlyphBBXOffX(u8g->font, g);

buf->w = - u8g->glyph\_x;

/\* Also copy the position of the first glyph. This is the reference point of the string (negated) \*/

buf->x = u8g->glyph\_x;

buf->y = u8g->glyph\_y;

for(;;)

{

/\* calculated y position of the upper left corner (y\_max) and lower left corner (y\_min) of the string \*/

/\* relative to the base line \*/

tmp = u8g->glyph\_y;

if ( buf->y\_min > tmp )

buf->y\_min = tmp;

tmp +=u8g->glyph\_height;

if ( buf->y\_max < tmp )

buf->y\_max = tmp;

/\* check and stop if the end of the string is reached \*/

s++;

if ( \*s == '\0' )

break;

/\* if there are still more characters, add the delta to the next glyph \*/

buf->w += u8g->glyph\_dx;

/\* load the next glyph information \*/

u8g\_GetGlyph(u8g, \*s);

}

/\* finally calculate the width of the last char \*/

/\* if g was not updated in the for loop (strlen() == 1), then the initial offset x gets removed \*/

buf->w += u8g->glyph\_width;

// buf->w += u8g\_font\_GetGlyphBBXOffX(u8g->font, g);

buf->w += u8g->glyph\_x;

}

/\* calculate minimal box \*/

void u8g\_font\_box\_min(u8g\_t \*u8g, const char \*s, u8g\_str\_size\_t \*buf)

{

u8g\_font\_calc\_str\_min\_box(u8g, s, buf);

}

/\* calculate gA box, but do not calculate the overall width \*/

void u8g\_font\_box\_left\_gA(u8g\_t \*u8g, const char \*s, u8g\_str\_size\_t \*buf)

{

}

/\* calculate gA box, including overall width \*/

void u8g\_font\_box\_all\_gA(u8g\_t \*u8g, const char \*s, u8g\_str\_size\_t \*buf)

{

}

static void u8g\_font\_get\_str\_box\_fill\_args(u8g\_t \*u8g, const char \*s, u8g\_str\_size\_t \*buf, u8g\_uint\_t \*x, u8g\_uint\_t \*y, u8g\_uint\_t \*width, u8g\_uint\_t \*height)

{

/\*

u8g\_glyph\_t g;

g =

\*/

u8g\_GetGlyph(u8g, \*s);

\*x += u8g->glyph\_x;

\*width = buf->w;

\*y -= buf->y\_max;

/\* +1 because y\_max is a height, this compensates the next step \*/

//\*y += 1;

/\* because the reference point is one below the string, this compensates the previous step \*/

//\*y -= 1;

\*height = buf->y\_max;

\*height -= buf->y\_min;

}

void u8g\_GetStrMinBox(u8g\_t \*u8g, const char \*s, u8g\_uint\_t \*x, u8g\_uint\_t \*y, u8g\_uint\_t \*width, u8g\_uint\_t \*height)

{

u8g\_str\_size\_t buf;

if ( \*s == '\0' )

{

\*width= 0;

\*height = 0;

return;

}

u8g\_font\_calc\_str\_min\_box(u8g, s, &buf);

u8g\_font\_get\_str\_box\_fill\_args(u8g, s, &buf, x, y, width, height);

}

void u8g\_GetStrAMinBox(u8g\_t \*u8g, const char \*s, u8g\_uint\_t \*x, u8g\_uint\_t \*y, u8g\_uint\_t \*width, u8g\_uint\_t \*height)

{

u8g\_str\_size\_t buf;

uint8\_t cap\_a;

if ( \*s == '\0' )

{

\*width= 0;

\*height = 0;

return;

}

cap\_a = u8g\_font\_GetCapitalAHeight(u8g->font);

u8g\_font\_calc\_str\_min\_box(u8g, s, &buf);

if ( buf.y\_max < cap\_a )

buf.y\_max = cap\_a;

u8g\_font\_get\_str\_box\_fill\_args(u8g, s, &buf, x, y, width, height);

}

void u8g\_SetFont(u8g\_t \*u8g, const u8g\_fntpgm\_uint8\_t \*font)

{

if ( u8g->font != font )

{

u8g->font = font;

u8g\_UpdateRefHeight(u8g);

u8g\_SetFontPosBaseline(u8g);

}

}

/\*========================================================================\*/

/\* anti aliasing fonts \*/

int8\_t u8g\_draw\_aa\_glyph(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

const u8g\_pgm\_uint8\_t \*data;

uint8\_t w, h;

uint8\_t i, j;

u8g\_uint\_t ix, iy;

{

u8g\_glyph\_t g = u8g\_GetGlyph(u8g, encoding);

if ( g == NULL )

return 0;

data = u8g\_font\_GetGlyphDataStart(u8g->font, g);

}

w = u8g->glyph\_width;

h = u8g->glyph\_height;

x += u8g->glyph\_x;

y -= u8g->glyph\_y;

y--;

if ( u8g\_IsBBXIntersection(u8g, x, y-h+1, w, h) == 0 )

return u8g->glyph\_dx;

/\* now, w is reused as bytes per line \*/

w += 3;

w /= 4;

iy = y;

iy -= h;

iy++;

for( j = 0; j < h; j++ )

{

ix = x;

for( i = 0; i < w; i++ )

{

u8g\_Draw4TPixel(u8g, ix, iy, 0, u8g\_pgm\_read(data));

data++;

ix+=4;

}

iy++;

}

return u8g->glyph\_dx;

}

int8\_t u8g\_DrawAAGlyph(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t encoding)

{

y += u8g->font\_calc\_vref(u8g);

return u8g\_draw\_aa\_glyph(u8g, x, y, encoding);

}

u8g\_uint\_t u8g\_DrawAAStr(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, const char \*s)

{

u8g\_uint\_t t = 0;

int8\_t d;

if ( u8g\_font\_GetFormat(u8g->font) != 2 )

return 0;

//u8g\_uint\_t u8g\_GetStrWidth(u8g, s);

//u8g\_font\_GetFontAscent(u8g->font)-u8g\_font\_GetFontDescent(u8g->font);

y += u8g->font\_calc\_vref(u8g);

while( \*s != '\0' )

{

d = u8g\_draw\_aa\_glyph(u8g, x, y, \*s);

x += d;

t += d;

s++;

}

return t;

}