

Herptronix

Smart display dev. guide

DRAFT / UNCOMPLETE

DRAFT

1. Summary

1.	Summary.....	1
2.	Licensing	4
3.	Overview.....	5
3.1.	Introduction.....	5
3.2.	Pin description.....	5
3.3.	Schematics.....	6
3.4.	PCB.....	7
3.5.	Operating ratings.....	7
3.6.	Absolute maximum ratings	7
3.7.	Mode of operation	8
3.7.1.	Slave (default).....	8
3.7.2.	Embedded application.....	8
3.7.3.	Setup.....	10
3.8.	Software architecture.....	11
3.8.1.	Overview.....	11
3.8.2.	Files description.....	13
4.	Built-in resources.....	17
4.1.	Images	17
4.2.	Font.....	21
5.	P2D	23
5.1.	Overview.....	23
5.1.1.	Screen representation	23
5.1.2.	Variables type & enumerations.....	23
5.1.3.	Interface with the TFT	24
5.2.	Basic functions.....	25
5.2.1.	Initialization	25
5.2.2.	Get LCD dimension	25
5.2.3.	Colors.....	26
5.2.4.	Clear a surface	27
5.2.5.	Clipping.....	28
5.2.6.	Put a Pixel	29
5.2.7.	Draw a line.....	30
5.2.8.	Draw a rectangle	30
5.3.	CLUTs.....	30
5.3.1.	Introduction.....	30
5.3.2.	CLUT file description.....	31
5.3.3.	CLUT creation	31
5.3.4.	CLUT update	32
5.3.5.	CLUT configuration flags.....	32
5.4.	Text.....	33

5.4.1.	Font file description.....	33
5.4.2.	Set a font	34
5.4.3.	Display some text	34
5.4.4.	Get text dimension	35
5.5.	Sprites.....	35
5.5.1.	Sprite file description	36
5.5.2.	Set a CLUT	36
5.5.3.	Get sprite dimension	36
5.5.4.	Display a sprite	36
5.6.	Geometrics	37
5.6.1.	Draw a circle	37
5.6.2.	Draw a polygon.....	38
5.6.3.	Math functions	40
5.7.	Utils.....	41
5.8.	Internal buffering	42
5.8.1.	Create an internal surface	42
5.8.2.	Set the destination	43
5.8.3.	Copy a surface to screen	43
5.8.4.	Delete an internal surface	43
5.8.5.	Example	43
6.	GUI.....	44
6.1.	Overview.....	44
6.1.1.	Generic object	44
6.1.2.	Object attributes	44
6.1.3.	Group.....	46
6.1.4.	Signals.....	47
6.1.5.	Page creation / page handling / page jump	48
6.2.	Widgets.....	48
6.2.1.	Button.....	48
6.2.2.	Checkbox	49
6.2.3.	Radio.....	49
6.2.4.	Frame.....	50
6.2.5.	Text area.....	50
6.2.6.	Graph.....	51
6.2.7.	Led	52
6.2.8.	List	52
6.2.9.	Slider	52
6.2.10.	Value box.....	53
6.2.11.	Dial.....	53
6.2.12.	Value box (special Dial)	53
6.2.13.	User entry	53
6.3.	Macros.....	53

6.3.1.	Keyboard.....	53
6.3.2.	Popup.....	53
6.3.3.	Lists (macro)	53
6.3.4.	Files browser	53
7.	SD card.....	54
8.	RAM memory organization	54
8.1.	Memory allocator	54
8.2.	Allocable memory organization	55
8.3.	Serialized Shared Memory	55
9.	UART driving.....	58
9.1.	Message.....	58
9.2.	User message (from master to smart TFT).....	59
9.3.	Answer message (from smart TFT to master)	59
9.4.	Serialized commands.....	59
10.	Portability	68

2. Licensing

Software of this project is under GPL3 license:

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

DRAFT

3. Overview

3.1. Introduction

This smart TFT module is based on common and easy-to-find devices: a PIC32 drives a TFT display (3.2", ILI932X controller). The module is powered by a 5V power supply (supplied by the user) and provides some GPIO like UART, analog inputs, SPI and so on. The following pictures summarizes the architecture:

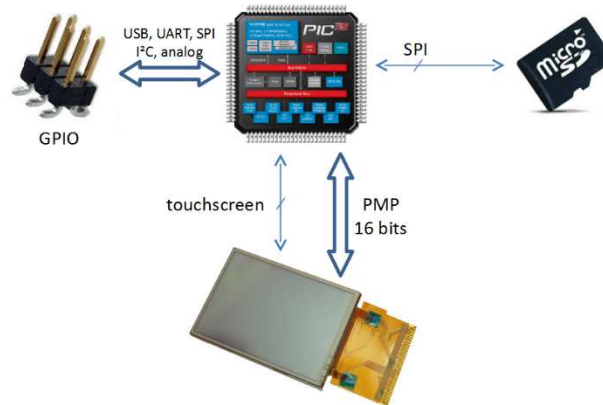


Figure 1: hardware architecture

Unlike most commercial smart displays, this one is open source (both hardware & software).

3.2. Pin description

This smart TFT comes with several I/O pins, coming directly from the PIC32. Refer to §3.3 for schematics.

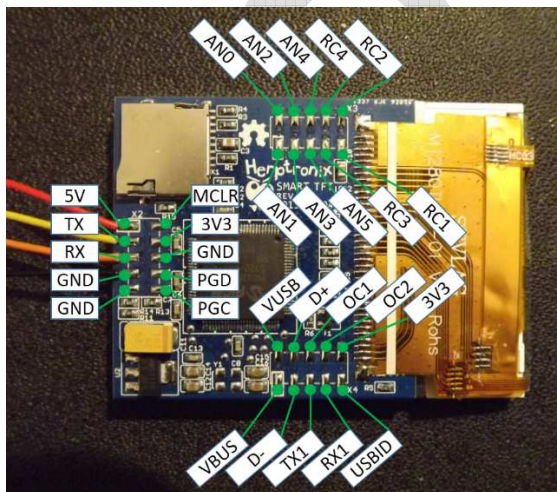


Figure 2: Pins description

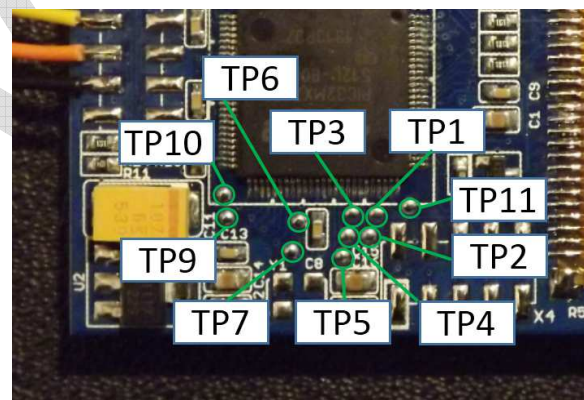
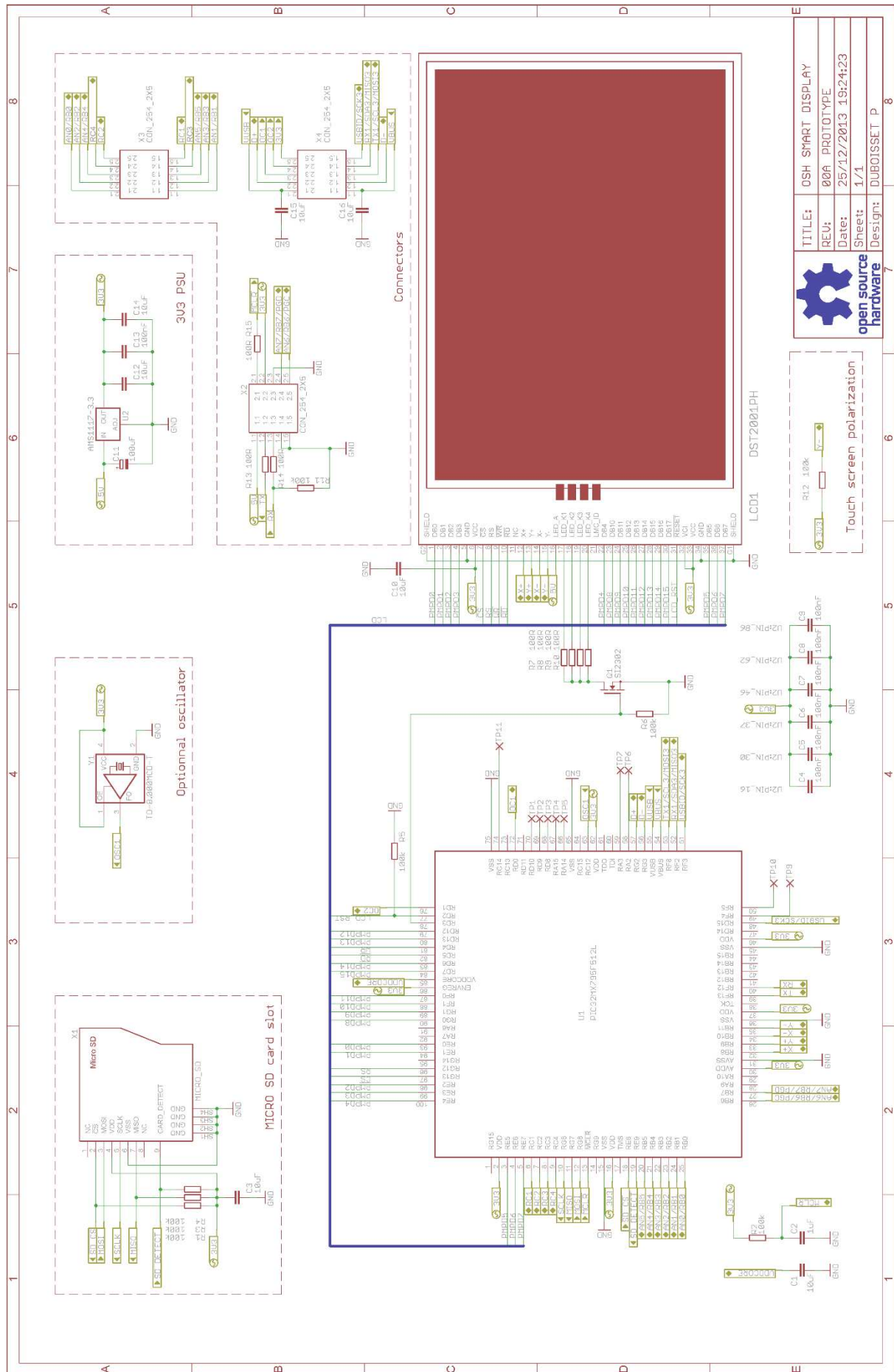


Figure 3: Test points description

3.3. Schematics



TITLE:	OSH SMART DISPLAY
REV:	00A PROTOTYPE
Date:	25/12/2013 19:24:23
Sheet:	1/1
Design:	DUBOISSET P

Touch screen polarization

R12 10k

3.4. PCB

The PCB is a 2 layers, 4.9 x 4.9 cm, with easy-to-process specifications (minimum wire length = 0.15mm, minimal drill diameter = 0.3mm...)

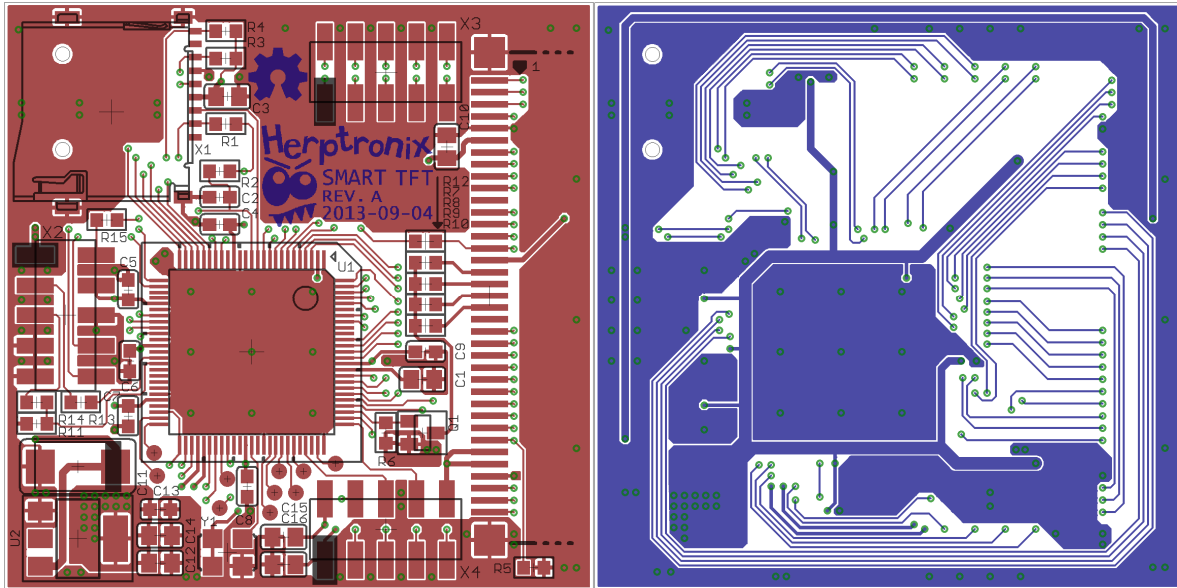


Figure 4: Top & bottom layers

3.5. Operating ratings

Table 1: operating ratings

Symbol	min	typ.	max	unit
(Vdd) Power supply voltage	4.5	5	5.5	V
(Idd) Current consumption ⁽¹⁾	130 ⁽²⁾	-	190 ⁽³⁾	mA
(Vin) Digital input voltage, 3.3V tolerant	0	-	3.3	V
(Vin) Digital input voltage, 5V tolerant	0	-	5	V
(Iout) Digital output current (3.3 & 5V pins)	0	-	20	mA

⁽¹⁾ Without SD card, no load on outputs, no ICSP probe

(2) During a minimalist page handling

⁽³⁾ During page switching (100% CPU activity, 10 to 300 ms peak)

3.6. Absolute maximum ratings

Table 2: absolute maximum ratings

Symbol	min	max	unit
(Vdd) Maximum power supply voltage	-0.2	6.0	V
(Vin) Digital input voltage, 3.3V tolerant	-0.2	3.6	V
(Vin) Digital input voltage, 5V tolerant	-0.2	5.5	V
(Ta) Ambient temperature	0	55	°C
Storage temperature	-20	70	°C

Unguaranteed / untested values

3.7. Mode of operation

This smart TFT has 2 different operational modes, selected at compilation time:

- Slave mode
- Embedded application mode

Another mode (setup mode) is always included, which provides a software demonstration, touchscreen configuration and gamma adjustment.

3.7.1. Slave (default)

In *slave* mode (which is the default mode), the smart TFT executes commands sent by a *master* (e.g. another MCU) through the UART. The firmware of the smart TFT does not require any modification, but the *master* shall integrate some additional source files for driving it. These source files are provided as an example, and should be re-written in a more efficient way.

For using the slave mode, the smart TFT shall be wired to a remote system through 4 wires:

- 5VIN: 5V power supply (see Table 1: operating ratings for current consumption)
- GND: power supply ground
- TX: Transmitted data (from the smart TFT to the master). This pin shall be connected to the RX of the master
- RX: Received data (from the master to the smart TFT). This pin shall be connected to the TX of the master



Default configuration is set to 1M bauds, 8N1. UART speed can be changed by modifying the `UART_SPEED` definition in `<src/app/serial_remote/serial_common.h>` file.

3.7.2. Embedded application

In this mode, the user application is integrated into the firmware of the smart TFT; user will have to modify the source code and upload its custom firmware into the smart TFT with an ICSP probe (e.g. a PICKIT 3). Paragraphs below describe a typical procedure for enabling this mode.



The smart TFT does not include a bootloader yet. An ICSP probe is required to upgrade the firmware

3.7.2.1. Get the MPLAB X project

The distributed MPLAB X project has been released using the following tools:

- MPLAB X version 1.90
- C32 compiler version 2.02, using `-O3` optimizations (if you do not have the license, the free version provides these optimizations during 60 days). Compilation with another compiler (e.g. XC32) has not been tested.

3.7.2.2. Configure the ICSP probe

Just ensure that the ICSP probe is powered by the smart TFT (here, a PICKIT3 cannot power the smart TFT; it would damage it):

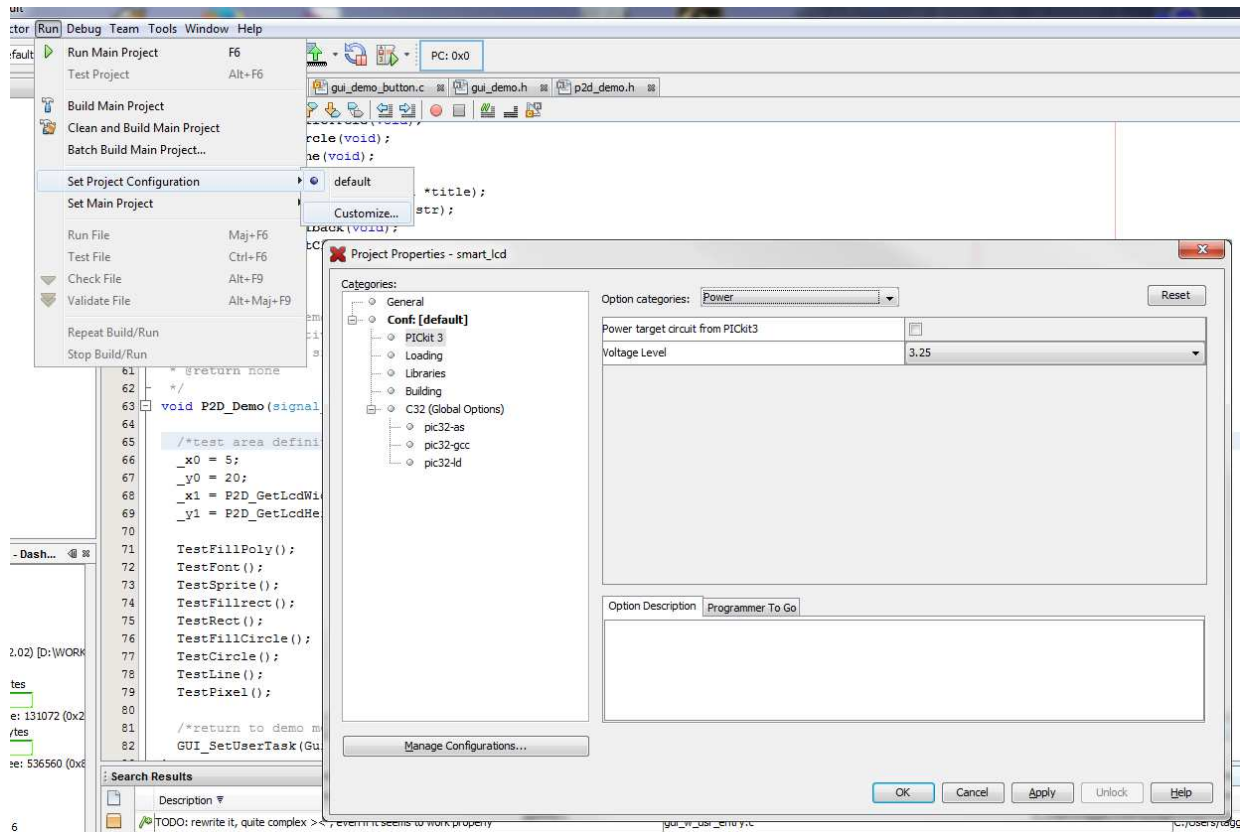


Figure 5: PICKIT3 configuration

3.7.2.3. Connecting the ICSP probe

A pin-to-pin header (compatible PICKIT3) is available:

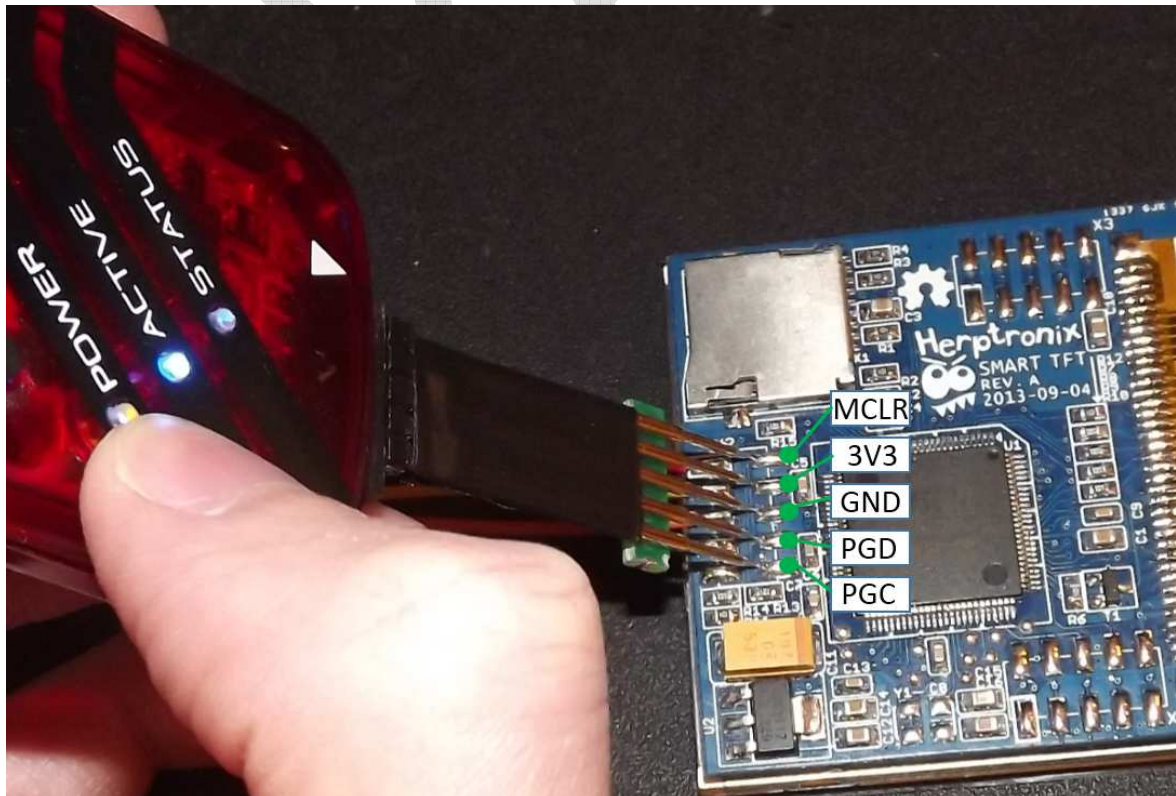


Figure 6: connecting the PICKIT3

3.7.2.4. Define the embedded app mode

Open the <src/main.h> header and comment the SMART_TFT_SLAVE_MODE symbol:

```
34  /**
35  * SMART_TFT_SLAVE_MODE
36  * Comment this flag if you want to embed your application directly on the smart TFT.
37  * Otherwise, the smart TFT executes orders coming from the serial comm
38  */
39  // #define SMART_TFT_SLAVE_MODE
```

3.7.2.5. Include your app

You can add your program in the UserTask() function, located in <src/app/usr/usr_main.c> file.

The firmware of the smart TFT will call this function at each software cycle.

A basic embedded application is included in source code (comes as an example).



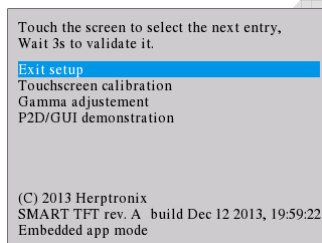
Due to the absence of RTOS, the embedded application shall manually yield the CPU (i.e. do not use infinite loop).

3.7.3. Setup

This mode allows configuring the touchscreen and running a demonstration. This mode is launched following the steps below:

- With the smart TFT powered OFF, press the touch screen
- By keeping the touch screen pressed, power ON the smart TFT

The setup screen appears:



After 3 seconds of inactivity from the touchscreen, the smart TFT will enter the highlighted item.

3.7.3.1. Touchscreen calibration



Touchscreen calibration is based on a 2 points algorithm



Parameters for touchscreen calibration are stored on the μ SD card (/TOUCHSCR.CFG file); if the μ SD card is not inserted, touchscreen will be calibrated with internal default parameters (i.e. without the TOUCHSCR.CFG file, the touchscreen may be “reversed”).

◆ Press the red point during 3 seconds

Press the red point during 3 seconds ◆

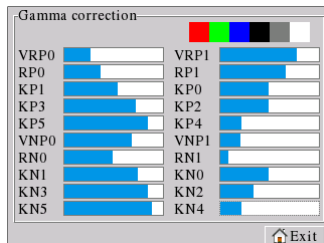
3.7.3.2. Gamma adjustment



Parameters for gamma adjustment are stored on the μ SD card (/GAMMA.CFG file); if the μ SD card is not inserted, gamma will be calibrated with internal default parameters



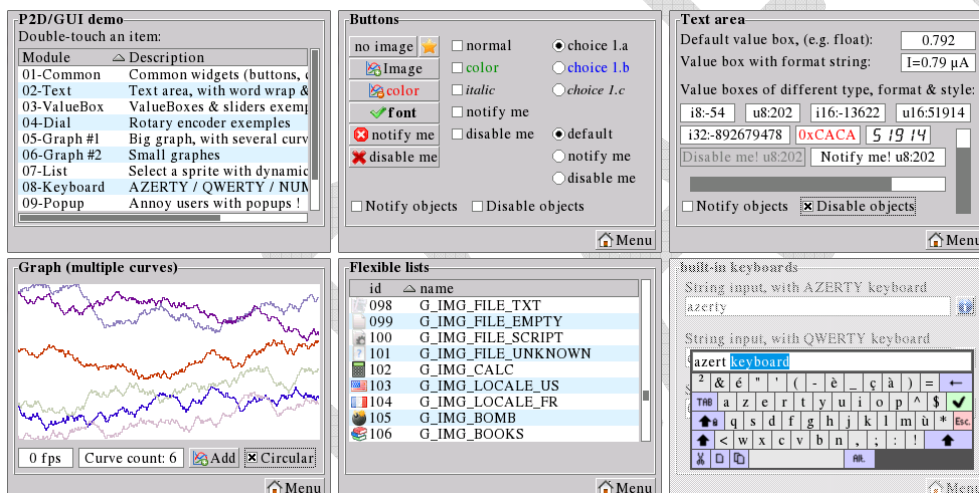
Report to the TFT controller documentation for the description of gamma registers



3.7.3.3. Demo



For reducing memory footprint, source code relative to the demonstration can be easily excluded from the compilation by commenting the `INCLUDE_GUI_DEMO` flag in `<main.h>`



3.8. Software architecture

3.8.1. Overview

The software is split in 3 parts: APP (the applicative part), SYS (FAT32 stack, memory allocator, etc...) and DRV (peripherals & microcontroller drivers).

The *Figure 7: software architecture* describes the software architecture:

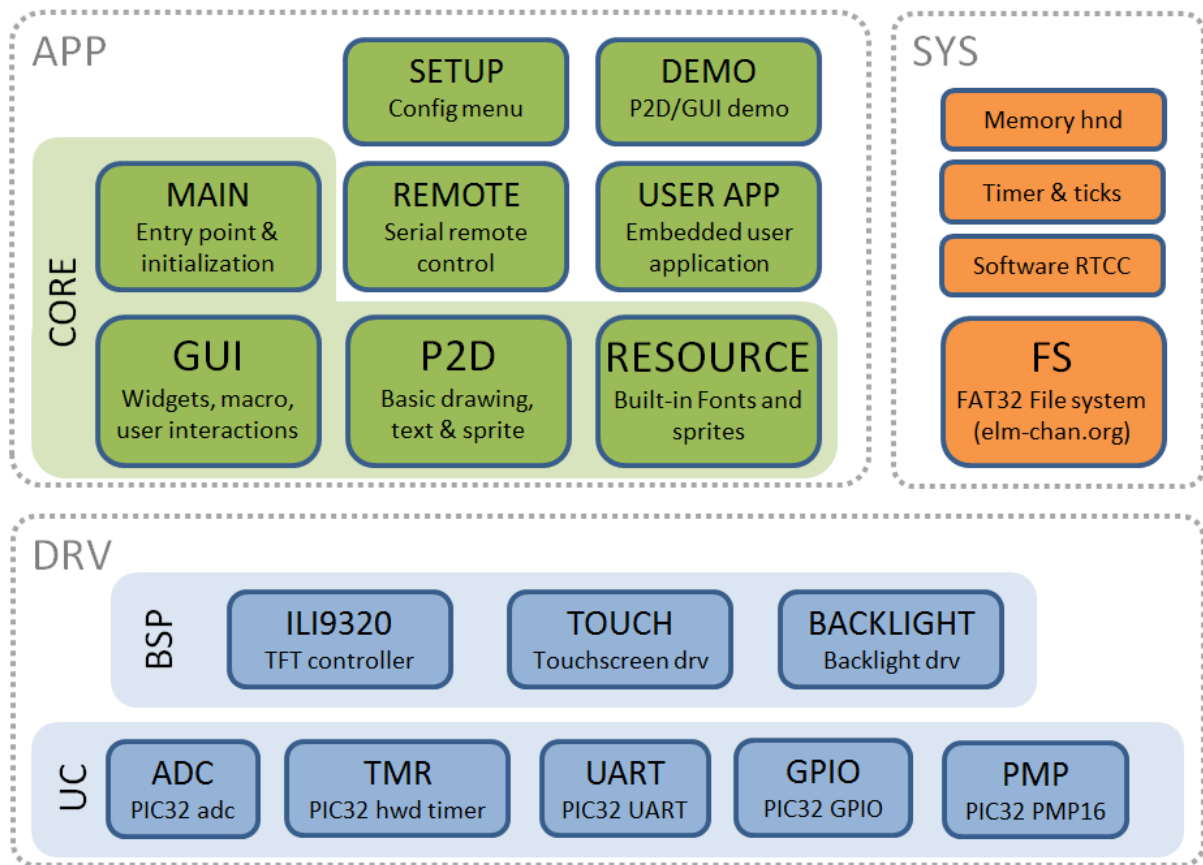


Figure 7: software architecture

3.8.2. Files description

The table above lists source code files constituting this project.

Table 3: Files description

File (.c)	Description
main	The main function; includes initialization and main loop
app\gui\macro\file_browser\gui_m_file_browser	Built-in file browser (μSD file browser)
app\gui\macro\keyboard\ gui_m_keyboard	Built-in keyboard (AZERTY / QWERTY & NUM, with insert / copy-paste features)
app\gui\macro\keyboard\ gui_w_key	Small widget for the built-in keyboard
app\gui\macro\keyboard\ keymap	Keyboards mappings (describes keys for AZERTY / QWERTY & NUM configuration)
app\gui\macro\list\ gui_m_list	Lists builder; create flexible lists, the <i>easy</i> way.
app\gui\macro\popup\ gui_m_popup	Built-in popups
app\gui\widgets\gui_w_button	Standard buttons
app\gui\widgets\gui_w_checkBox	Standard check boxes
app\gui\widgets\gui_w_frame	Frames (a text plus a rectangle)
app\gui\widgets\gui_w_graph	Graphs (normal & scrolled)
app\gui\widgets\gui_w_led	LED (red or green, according to their relative status)
app\gui\widgets\gui_w_list	Flexible lists implementation; use functions provided by gui_m_list
app\gui\widgets\gui_w_radio	Standard radios
app\gui\widgets\gui_w_rot_button	Rotary encoder
app\gui\widgets\gui_w_rot_value	Special value box, for rotary encoder
app\gui\widgets\gui_w_scroller	Standard scroll bar
app\gui\widgets\gui_w_slider	Standard slider
app\gui\widgets\gui_w_tab	Tabs (deprecated)
app\gui\widgets\gui_w_text	Text area (supports alignment & line feed)
app\gui\widgets\gui_w_usr_entry	User entry bar; allow string edition
app\gui\widgets\gui_w_valueBox	Value boxes; display formatted values

app\gui\gui_debug	GUI debug utils (memory usage)
app\gui\gui_graphics	GUI graphic wrapper; handle fonts & sprites through #id
app\gui\gui_obj	GUI generic object implementation + GUI task
app\gui\gui_utils	GUI utility
app\gui_demo\gui_demo	Main menu of GUI demo
app\gui_demo\gui_demo_button	generic widgets demo (buttons, radios, checkboxes)
app\gui_demo\gui_demo_file_browser	file browser demo
app\gui_demo\gui_demo_graph1	graph demo #1 (big graph with multiple curves)
app\gui_demo\gui_demo_graph2	graph demo #2 (small graphs & grid)
app\gui_demo\gui_demo_keyboard	keyboard demo
app\gui_demo\gui_demo_list	flexible lists demo
app\gui_demo\gui_demo_popup	popup demo
app\gui_demo\gui_demo_rbutton	rotary encoder demo
app\gui_demo\gui_demo_sd	SD card example
app\gui_demo\gui_demo_text	text area demo
app\gui_demo\gui_demo_valuebox	value boxes demo
app\p2d\p2d_base	Basic 2D functions (pixel, rectangle, ...)
app\p2d\p2d_buffer	Memory buffer (deprecated)
app\p2d\p2d_clip	Clipping
app\p2d\p2d_colors	Color conversion (RGB888 <-> RGB565, alpha, B&W...)
app\p2d\p2d_font	Text rendering
app\p2d\p2d_geo_circle	Circle & filled circle
app\p2d\p2d_geo_line	Lines
app\p2d\p2d_geo_poly	Polygon & filled polygon, plus polygon-oriented functions
app\p2d\p2d_internal	Functions dispatching (to TFT or to memory)
app\p2d\p2d_lut	Color look-up tables
app\p2d\p2d_math	Some math functions
app\p2d\p2d_sprite	Sprite rendering (only 8BPP)

app\p2d\p2d_utils	Some utils (distance calculation, etc...)
app\resources\FontDigit_4bpp	Font digit (7 segments style)
app\resources\FontSerif_4bpp_b	Font free serif, bold; comes with many heights.
app\resources\FontSerif_4bpp_i	Font free serif, italic; comes with many heights.
app\resources\FontSerif_4bpp_n	Font free serif, normal; comes with many heights.
app\resources\FontSymbol	Some symbol used by the built-in keyboard
app\resources\sprite00	Sprite pack #00 (16x16)
app\resources\sprite01	Sprite pack #01 (16x16)
app\resources\sprite02	Sprite pack #02 (16x16)
app\resources\sprite03	Sprite pack #03 (16x16)
app\resources\sprite04	Sprite pack #04 (32x32)
app\serial_remote\opList	List of operators (~ link with P2D/GUI) handled by the serial remote
app\serial_remote\r_gui	GUI serialized functions
app\serial_remote\r_mem	Memory serialized functions
app\serial_remote\r_p2d	P2D serialized functions
app\serial_remote\serial_remote	Serial remote task & FSM
app\serial_remote\serial_rx	Functions for message reception
app\serial_remote\serial_tx	Functions for message emission
app\serial_remote\ssm	Serialized shared memory
app\setup\gamma	Gamma adjustment
app\setup\setup	Setup menu
app\setup\touchCalib	Touchscreen calibration
app\user_app\usr_main	User application (disabled by default)
drv\bsp\backlight	Backlight driver (dummy for now)
drv\bsp\ILI9320	TFT controller driver
drv\bsp\touchscreen	Touchscreen driver
drv\uc\adc	PIC32 ADC driver
drv\uc\configuration_bits	PIC32 fuses

drv\uc\gpio	PIC32 GPIO driver
drv\uc\hw_config	Hardware constants (e.g. clock frequency)
drv\uc\pmp	PIC32 parallel master port driver (16 bits mode)
drv\uc\sem	Atomic write macro (quick & dirty spinlock for interrupts)
drv\uc\tmr	PIC32 timers driver
drv\uc\uart	PIC32 UART driver
drv\uc\uc	PIC32 core utils (e.g. sleep mode)
sys\file_system\diskio	FAT-FS part (see http://elm-chan.org/fsw/ff/00index_e.html)
sys\file_system\ff	FAT-FS part (see http://elm-chan.org/fsw/ff/00index_e.html)
sys\file_system\ffconf	FAT-FS part (see http://elm-chan.org/fsw/ff/00index_e.html)
sys\file_system\integer	FAT-FS part (see http://elm-chan.org/fsw/ff/00index_e.html)
sys\file_system\mmcPIC32	PIC32 specific for FAT-FS
sys\delay	Delay functions
sys\rtc	Software RTCC (needed by FAT-FS)
sys\salloc	Custom memory allocator (used by GUI)
sys\ticks	Software clock
sys\timer	Software timers

4. Built-in resources

The firmware includes some built-in graphical resources (images and fonts). These resources are listed in the tables below.

4.1. Images

The tables below list built-in images and their available CLUTs. Built-in images come from the open icon library project (<http://openiconlibrary.sourceforge.net/>)




Due to bandwidth limitations, P2D & GUI do not handle images which are located on the micro SD card.





























Table 4: available CLUTs

CLUT symbol	Description
G_LUT_NORMAL	This CLUT is the 'default' one; by using this CLUT, the image will be displayed as it appears in <i>Table 5</i> & <i>Table 6</i>
G_LUT_DISABLED	By using this CLUT, the image will be displayed in black & white; due to memory limitations, this CLUT is not available for all images.

Table 5: built-in 16x16 images

	defined symbol (see src/app/resources.h)	Available CLUTs
✓	G_IMG_OK	G_LUT_NORMAL, G_LUT_DISABLED
✗	G_IMG_NO	G_LUT_NORMAL, G_LUT_DISABLED
+	G_IMG_ADD	G_LUT_NORMAL, G_LUT_DISABLED
−	G_IMG_REMOVE	G_LUT_NORMAL, G_LUT_DISABLED
←	G_IMG_LEFT_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
↶	G_IMG_LEFT_D_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
↓	G_IMG_DOWN_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
↷	G_IMG_DOWN_D_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
→	G_IMG_RIGHT_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
↘	G_IMG_RIGHT_D_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
↑	G_IMG_UP_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
↖	G_IMG_UP_D_ARROW	G_LUT_NORMAL, G_LUT_DISABLED
⬅	G_IMG_GO_LEFT	G_LUT_NORMAL, G_LUT_DISABLED
⬇	G_IMG_GO_DOWN	G_LUT_NORMAL, G_LUT_DISABLED
➡	G_IMG_GO_RIGHT	G_LUT_NORMAL, G_LUT_DISABLED
⬆	G_IMG_GO_UP	G_LUT_NORMAL, G_LUT_DISABLED
⚙	G_IMG_CONFIG	G_LUT_NORMAL, G_LUT_DISABLED
🚫	G_IMG_FORBID	G_LUT_NORMAL, G_LUT_DISABLED
❌	G_IMG_ERROR	G_LUT_NORMAL, G_LUT_DISABLED
💾	G_IMG_SAVE	G_LUT_NORMAL, G_LUT_DISABLED
📄	G_IMG_OPEN	G_LUT_NORMAL, G_LUT_DISABLED
✎	G_IMG_EDIT	G_LUT_NORMAL, G_LUT_DISABLED

	G_IMG_INFO	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_HELP	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_HOME	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_REFRESH	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_IMPORTANT	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_EXIT	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_EJECT	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_PAUSE	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_STOP	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_REC	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_FIND	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_ZOOM_1	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_ZOOM_OUT	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_ZOOM_IN	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_ACCEPT	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_DECLINE	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_SORT_DOWN	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_SORT_UP	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_BOOKMARK	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_GRAPH_ADD	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_GRAPH_REMOVE	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_GRAPH_EDIT	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_DOWNLOAD	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_CLEAR	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_CUT	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_DRAG	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_SMILE	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_HISTOGRAM	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_LOCK	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_BULB	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_OSCILLOSCOPE	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_TASK	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_WARNING	G_LUT_NORMAL , G_LUT_DISABLED
	G_IMG_DEVICE	G_LUT_NORMAL
	G_IMG_SND_MUTE	G_LUT_NORMAL
	G_IMG_SOUND_MIN	G_LUT_NORMAL
	G_IMG_SOUND_MEDIUM	G_LUT_NORMAL
	G_IMG_SOUND_MAX	G_LUT_NORMAL
	G_IMG_BATTERY_MIN	G_LUT_NORMAL
	G_IMG_BATTERY_MEDIUM	G_LUT_NORMAL
	G_IMG_BATTERY_GOOD	G_LUT_NORMAL
	G_IMG_BATTERY_MAX	G_LUT_NORMAL
	G_IMG_BATTERY_C_MIN	G_LUT_NORMAL
	G_IMG_BATTERY_C_MEDIUM	G_LUT_NORMAL
	G_IMG_BATTERY_C_GOOD	G_LUT_NORMAL

	G_IMG_BATTERY_C_MAX	G_LUT_NORMAL
	G_IMG_COMPUTER	G_LUT_NORMAL
	G_IMG_SD_CARD	G_LUT_NORMAL
	G_IMG_SIGNAL_NONE	G_LUT_NORMAL
	G_IMG_SIGNAL_MIN	G_LUT_NORMAL
	G_IMG_SIGNAL_MEDIUM	G_LUT_NORMAL
	G_IMG_SIGNAL_GOOD	G_LUT_NORMAL
	G_IMG_SIGNAL_MAX	G_LUT_NORMAL
	G_IMG_SAFE	G_LUT_NORMAL
	G_IMG_UNSAFE	G_LUT_NORMAL
	G_IMG_USB	G_LUT_NORMAL
	G_IMG_CLOCK	G_LUT_NORMAL
	G_IMG_FILE_INFO	G_LUT_NORMAL
	G_IMG_FILE_APP	G_LUT_NORMAL
	G_IMG_FILE_APP_WIN	G_LUT_NORMAL
	G_IMG_FILE_ZIP	G_LUT_NORMAL
	G_IMG_FILE_AUDIO	G_LUT_NORMAL
	G_IMG_FILE_PACK	G_LUT_NORMAL
	G_IMG_FILE_ENCRYPTED	G_LUT_NORMAL
	G_IMG_FILE_IMPORTANT	G_LUT_NORMAL
	G_IMG_FILE_TMP	G_LUT_NORMAL
	G_IMG_FOLDER	G_LUT_NORMAL
	G_IMG_FILE_FONT	G_LUT_NORMAL
	G_IMG_FILE_IMGS	G_LUT_NORMAL
	G_IMG_FILE_IMG	G_LUT_NORMAL
	G_IMG_FILE_PACK2	G_LUT_NORMAL
	G_IMG_FILE_HTML	G_LUT_NORMAL
	G_IMG_FILE_ASM	G_LUT_NORMAL
	G_IMG_FILE_H	G_LUT_NORMAL
	G_IMG_FILE_C	G_LUT_NORMAL
	G_IMG_FILE_TXT	G_LUT_NORMAL
	G_IMG_FILE_EMPTY	G_LUT_NORMAL
	G_IMG_FILE_SCRIPT	G_LUT_NORMAL
	G_IMG_FILE_UNKNOWN	G_LUT_NORMAL
	G_IMG_CALC	G_LUT_NORMAL
	G_IMG_LOCALE_US	G_LUT_NORMAL
	G_IMG_LOCALE_FR	G_LUT_NORMAL
	G_IMG_BOMB	G_LUT_NORMAL
	G_IMG_BOOKS	G_LUT_NORMAL
	G_IMG_PACKAGE	G_LUT_NORMAL
	G_IMG_BRICKS	G_LUT_NORMAL
	G_IMG_CART	G_LUT_NORMAL
	G_IMG_CREDIT_CARDS	G_LUT_NORMAL
	G_IMG_DB_ERROR	G_LUT_NORMAL
	G_IMG_EYE	G_LUT_NORMAL

































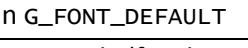
	G_IMG_FLAG_BLUE	G_LUT_NORMAL
	G_IMG_FLAG_GREEN	G_LUT_NORMAL
	G_IMG_FLAG_ORANGE	G_LUT_NORMAL
	G_IMG_FLAG_PINK	G_LUT_NORMAL
	G_IMG_FLAG_PURPLE	G_LUT_NORMAL
	G_IMG_FLAG_RED	G_LUT_NORMAL
	G_IMG_FLAG_YELLOW	G_LUT_NORMAL
	G_IMG_PICTURES	G_LUT_NORMAL
	G_IMG_WINDOWS	G_LUT_NORMAL
	G_IMG_MONEY	G_LUT_NORMAL
	G_IMG_MUSIC	G_LUT_NORMAL
	G_IMG_SHEETS	G_LUT_NORMAL
	G_IMG_PACKAGE2	G_LUT_NORMAL
	G_IMG_ROSETTE	G_LUT_NORMAL
	G_IMG_TAG_BLUE	G_LUT_NORMAL
	G_IMG_TAG_GREEN	G_LUT_NORMAL
	G_IMG_TAG_ORANGE	G_LUT_NORMAL
	G_IMG_TAG_PINK	G_LUT_NORMAL
	G_IMG_TAG_PURPLE	G_LUT_NORMAL
	G_IMG_TAG_RED	G_LUT_NORMAL
	G_IMG_TAG_YELLOW	G_LUT_NORMAL
	G_IMG_TUX	G_LUT_NORMAL
	G_IMG_USER_BLUE	G_LUT_NORMAL
	G_IMG_USER_PURPLE	G_LUT_NORMAL
	G_IMG_USER_GREY	G_LUT_NORMAL
	G_IMG_USER_GREEN	G_LUT_NORMAL
	G_IMG_USER_ORANGE	G_LUT_NORMAL
	G_IMG_USER_RED	G_LUT_NORMAL
	G_IMG_USER_BLUE2	G_LUT_NORMAL

Table 6: built-in 32x32 images

	defined symbol	Available CLUTs
	G_IMG_BIG_ERROR	G_LUT_NORMAL
	G_IMG_BIG_OK	G_LUT_NORMAL
	G_IMG_BIG_WARNING	G_LUT_NORMAL
	G_IMG_BIG_INFO	G_LUT_NORMAL
	G_IMG_BIG_ASK	G_LUT_NORMAL
	G_IMG_BIG_SAVED	G_LUT_NORMAL
	G_IMG_BIG_SD_CARD	G_LUT_NORMAL

	G_FONT_VERY_BIG	32	255
same than G_FONT_DEFAULT	G_FONT_SMALL	32	255
Internal usage only (for the keyboard)	G_FONT_SYMBOL	1	10

A lot of fonts is already available (but excluded from compilation due to memory limitation). To include / exclude a font, its corresponding symbol shall be commented / uncommented in file `<app/resources/resources.h>`:

Figure 8: font include / exclude

```
43 // #define USE_FONT_DIGIT_4BPP_N_31
44 // #define USE_FONT_DIGIT_4BPP_N_39
45
46 // #define USE_FONT_FREESERIF_4BPP_N_12
47 // #define USE_FONT_FREESERIF_4BPP_N_14
48 #define USE_FONT_FREESERIF_4BPP_N_16
49 // #define USE_FONT_FREESERIF_4BPP_N_18
50 #define USE_FONT_FREESERIF_4BPP_N_20
51 // #define USE_FONT_FREESERIF_4BPP_N_22
52 #define USE_FONT_FREESERIF_4BPP_N_24
```

Font name	Bit per pixel	N: normal B: Bold I: Italic	Font height (in pixels)
-----------	---------------	-----------------------------------	----------------------------

5. P2D



Some functions provided by P2D are not documented. These undocumented functions are mainly used by P2D itself.

5.1. Overview

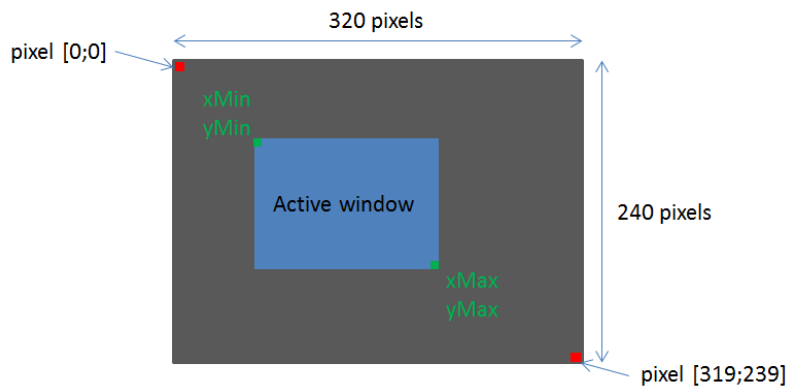
P2D (stands for *Primitives 2D*) is a set of functions which perform basic graphical operations (e.g. draw a line, display a text or an image, etc...).

P2D is written in C language only.

5.1.1. Screen representation

The smart TFT (rev. a) is based on a graphical controller (ILI932x) which drives a 320x240 TFT screen. For now, P2D handles only one display orientation; the figure above shows how pixels are identified:

Figure 9: screen representation



P2D needs some features provided by graphical controllers:

- Cursor placement: this feature allows P2D to place the cursor anywhere on the screen
- Cursor auto-increment: each time a pixel is put on the screen, the cursor is shifted to the right (x++); when reaching the right border, the cursor is automatically set to the left border of the next line (x = 0; y++)
- Hardware window: this feature allows to set the *left* and *right borders* used by the auto increment (delimited by *xMin*, *yMin*, *xMax*, *yMax* of Figure 9)

Other functionalities (e.g. hardware scrolling, hardware BitBlt or JPEG decompression provided by a SSD1926) are not used / supported.

5.1.2. Variables type & enumerations

Most P2D functions rely on specific data types listed below, and defined in `ILI9320.h` or `P2D_base.h` file:

Table 8: variables type

type	description
<code>coord_t</code>	Pixel coordinate (e.g. x or y) Equivalent to a 16bits signed integer; range [-32768 to 32767]

length_t	Length in pixel (i.e. length between 2 coordinates, in pixel. For x1 equal to x2, length is equal to 1) Equivalent to a 16 bits unsigned integer; range [0 to 65535]
typedef struct { coord_t x, y; length_t w, h; } rect_st;	Defines a rectangular surface, given by a point (x and y), a width w (the distance between xMax & xMin, in pixels) and a height h (the distance between yMax & yMin, in pixels). A valid rect_st shall fulfill $w \geq h \geq 1$
color_t	Color (standard RGB565 formatted). Equivalent to a 16 bits unsigned integer; 0x0000 ↔ black, 0xFFFF ↔ white
line_t	LINE_SOLID (draw a continuous line) LINE_DOT (draw a dotted line)
dmode_t	DISPLAY_SOLID DISPLAY_TRANSPARENT

5.1.3. Interface with the TFT

P2D is interfaced with few external functions, defined in `ILI9320.h` file, making easier the portability to another platform (see §10). These functions are listed below:

LCD_Init		
description	Initializes the TFT screen (registers setting) and turns it ON	
argument	none	
return	none	

LCD_Getwidth		
description	Returns the TFT width, in pixels	
argument	none	
return	length_t	here, width = 320 pixels

LCD_GetHeight		
description	Returns the TFT height, in pixels	
argument	none	
return	length_t	here, height = 240 pixels

LCD_Setwnd		
description	Defines the hardware windows	
argument	const rect_st *rec	new hardware window
return	none	

LCD_SetPos		
description	Sets the cursor position (i.e. the pixel which will be written next)	
argument	coord_t x, y	new cursor position; the cursor position shall always be within the hardware windows dimension
return	none	

LCD_Put		
description	Sets the current pixel to a given color, and increment the cursor position; the cursor increment is always contained within the hardware window	
argument	color_t col	RGB565 color
return	none	

All functions provided by P2D rely on these 6 elementary functions.

5.2. Basic functions



Some functions come with some source code examples. To run these examples, just copy/paste the given code in file <src/app/user_app/user_app.c> at line 34; Do not forget to set the operational mode to *embedded application* (§3.7.2)

5.2.1. Initialization

The given source code already initializes P2D and its dependent software component. Nevertheless, this initialization is performed through the following function:

P2D_Init		
description	Initializes the 2D primitives; Shall be called before using any P2D function. The following operations are performed: <ul style="list-style-type: none"> - Set destination to the physical screen - Set display mode as DISPLAY_SOLID - Set front/back colors to COLOR_BLACK - Set alpha to 255 (opaque) - Set line type to LINE_SOLID - Select the default font - Expand the clip to the screen dimensions - Clear the screen (COLOR_BLACK) 	
argument	none	
return	none	

5.2.2. Get LCD dimension

P2D_GetLcdwidth		
description	Returns the width (in pixel) of the screen	
argument	none	
return	length_t	width (in pixels) of the screen
example	<pre>length_t w; w = P2D_GetLcdwidth(); // w = 320 here</pre>	

P2D_GetLcdHeight		
description	Returns the height (in pixel) of the screen	
argument	none	
return	length_t	height (in pixels) of the screen
example	<pre>length_t h; h = P2D_GetLcdHeight(); // h = 240 here</pre>	

5.2.3. Colors

5.2.3.1. Format

As described in §5.1.2, the color is coded on 16 bits, following the RGB565 format, where the red channel is 5 bits width, green is 6 bits width and blue is 5 bits width:

R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

5.2.3.2. Colors conversion

Since it is not easy to write colors in this format, a function making the translation from RGB888 to RGB565 is available:

P2D_Color		
description	Converts a standard RGB 888 into a RGB 565	
argument	uint8_t r	red channel (0 to 255)
	uint8_t g	green channel (0 to 255)
	uint8_t b	blue channel (0 to 255)
return	color_t	RGB565 color
example	<pre>color_t color; color = P2D_Color(255, 0, 0); //red color color = P2D_Color(0, 255, 0); //green color color = P2D_Color(0, 0, 255); //blue color</pre>	

The complementary function (conversion from RGB565 to RGB888) is also available:

P2D_ColorToRGB		
description	Converts a RGB 565 into a RGB 888; introduces extrapolation errors!	
argument	color_t color	RGB565 color input
	uint8_t *r	red channel (0 to 255) output
	uint8_t *g	green channel (0 to 255) output
	uint8_t *b	blue channel (0 to 255) output
return	none	
example	<pre>uint8_t r, g, b; P2D_ColorToRGB(COLOR_GREEN, &r, &g, &b); /* normally, r should be = 0, g = 255, b = 0 * but due to extrapolation error, g will be != of 255 */</pre>	

P2D_ColorBlackAndWhite		
description	Turns a color into a black and white level	
argument	color_t a	input color
return	color_t	output color (black & white)
example	<pre>color_t color; color = P2D_ColorBlackAndWhite(COLOR_RED);</pre>	

P2D_Alpha_a_on_b		
description	Computes the color value corresponding to alpha(color A on color B)	
argument	color_t a	RGB565 color input
	color_t b	red channel (0 to 255) output
	uint8_t alpha	green channel (0 to 255) output

return	color_t	output color
example	<pre>color_t color; color = P2D_Alpha_a_on_b(COLOR_RED, COLOR_WHITE, 128);</pre>	

5.2.3.3. Define the color to use

Colors to use shall be defined before using a drawing function (e.g. if you want to draw a red line, you shall set the color first). P2D handles 2 different colors:

- The *front* color: this color is used by most drawing functions (e.g. draw a pixel, a line, a rectangle...)
- The *background* color: this color is used only some functions (e.g. draw a sprite, draw a text), and is mainly 'performance-oriented'.

P2D_SetColor		
description	Sets the front color to use for next operations	
argument	color_t front	new front color
return	none	
example	<pre>P2D_SetColor(COLOR_RED); P2D_Clear(); //fills the screen in red</pre>	

P2D_SetColors		
description	Sets the front color and the background color to use for next operations	
argument	color_t front	new front color
	color_t back	new background color
return	none	
example	<pre>P2D_SetColors(COLOR_RED, COLOR_BLACK); P2D_SetDisplayMode(DISPLAY_SOLID); P2D_PutText(0, 0, "fastest text rendering");</pre>	

5.2.3.4. Definitions

Some common colors are already defined in p2d_colors.h:

Table 9: defined colors

Symbol	R	G	B
COLOR_BLACK	0	0	0
COLOR_DARK_GREY	64	64	64
COLOR_GREY	128	128	128
COLOR_LIGHT_GREY	196	196	196
COLOR_WHITE	255	255	255
COLOR_RED	255	0	0
COLOR_ORANGE	255	128	0
COLOR_YELLOW	255	255	0
COLOR_GREEN	0	255	0
COLOR_CYAN	0	255	255
COLOR_BLUE	0	0	255
COLOR_PURPLE	255	0	255

5.2.4. Clear a surface

P2D_Clear

description	Clears the current surface <u>within the current clip</u> . The current surface may be the screen or an internal surface located memory (see §5.8 for internal surface, §5.2.5 for clipping)
argument	none
return	none
example	<pre>/*clear (in black) the whole screen*/ P2D_SetColor(COLOR_BLACK); P2D_SetClip(NULL); P2D_Clear(); /*clear (in white) a part of the screen*/ rect_st rec = {10, 20, 50, 50}; P2D_SetClip(&rec); P2D_SetColor(COLOR_WHITE); P2D_Clear();</pre>

5.2.5. Clipping

5.2.5.1. Introduction

The *clipping* module defines a display area; operations outside this area will not be performed. The above figure gives a typical example:

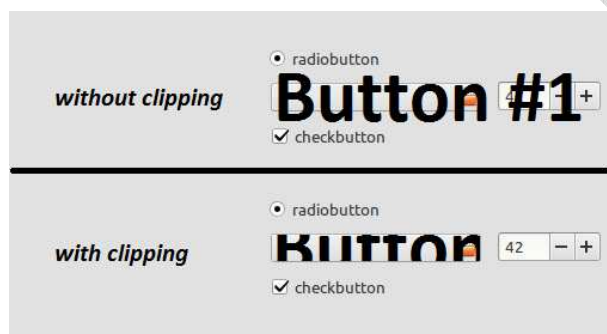


Figure 10: with / without clipping



Always configure the clip before putting anything on the screen (possible concurrent access with GUI)



All P2D drawing functions are delimited by the clipping.

5.2.5.2. Defining a clip

P2D_SetClip		
description	Defines the <i>global</i> clip. All the following operations will be contained into this clip.	
argument	const rect_st *rec	Clip to use; if NULL, the clip will be disabled (i.e. the clip is equal to the current surface dimension)
return	none	
example	<pre>/*disable the clip (~expand to surface physical dimension)*/ P2D_SetClip(NULL); /*set a clip*/ rect_st rec = {10, 20, 50, 50}; P2D_SetClip(&rec); /*now, all drawing functions cannot modify pixels which do not belong to the defined rec*/</pre>	

5.2.5.3. Operations on clip

P2D_IsCoordInClip		
description	Checks if a given point is within the global clip	
argument	coord_t x, y	Absolute coordinates of the given point
return	bool	if true, the point is within the clip
example	<pre>bool bIn; /*disable the clip (~expand to surface dimension)*/ P2D_SetClip(NULL); bIn = P2D_IsCoordInClip(1, 2); //true, [1;2] is within {0;0;320;240} /*set a clip*/ rect_st rec = {10, 20, 50, 50}; P2D_SetClip(&rec); bIn = P2D_IsCoordInClip(1, 2); //false, [1;2] is out of {10;20;50;50}</pre>	

P2D_Clip		
description	Modifies a given <i>src</i> clip in order to fit it in another given <i>dst</i> clip	
argument	rect_st *src	a random clip; will be modified (if necessary) to fit into the <i>dst</i> clip
	const rect_st *dst	the reference clip
return	none	
example	<pre>rect_st clipToFit = {0, 0, 320, 100}; rect_st clipRef_1 = {0, 0, 320, 240}; rect_st clipRef_2 = {20, 20, 300, 200}; P2D_Clip(&clipToFit, &clipRef_1); /*no change, clipToFit is smaller than clipRef_1*/ P2D_Clip(&clipToFit, &clipRef_2); /*clipToFit changed to {20, 20, 300, 100}*/</pre>	

P2D_ClipFit		
description	Modifies a given clip in order to fit it in the <i>global</i> clip. This is just a call to <i>P2D_Clip()</i> with the <i>global</i> clip as the <i>dst</i> argument.	
argument	rect_st *toClip	a random clip; will be modified (if necessary) to fit into the <i>global</i> clip
return	none	

5.2.6. Put a Pixel

P2D_SetPixel		
description	Puts a pixel on the current surface	
argument	coord_t x, y	Absolute coordinates of the given pixel
return	none	
example	P2D_SetPixel(10, 20, COLOR_RED); //put a red pixel at x=10, y=20	

Note: this function is not suitable for performance: for each pixel, this function checks if the given coordinates are within the *global* clip; as a consequence, this function cannot be used for achieving high performance drawing functions.

5.2.7. Draw a line

P2D_Line		
description	Draws a line on the current surface	
argument	coord_t x0, y0	Absolute coordinates of the given pixel
	coord_t x1, y1	
return	none	
example	<pre>P2D_SetColor(COLOR_RED); P2D_Line(10, 20, 110, 120); //red a red line between 10;20 and 110;120</pre>	

P2D_SetLineType		
description	Sets the type of line (continuous or dotted)	
argument	line_t type	LINE_SOLID for continuous line LINE_DOT for dotted line
return	none	
example	<pre>P2D_SetColor(COLOR_GREY); P2D_Clear(); P2D_SetLineType(LINE_SOLID); P2D_SetColor(COLOR_RED); P2D_Line(10, 10, 100, 10); P2D_SetColors(COLOR_BLACK, COLOR_WHITE); P2D_SetLineType(LINE_DOT); P2D_Line(10, 20, 100, 20); P2D_SetDisplayMode(DISPLAY_TRANSPARENT); P2D_Line(10, 30, 100, 30);</pre>	

5.2.8. Draw a rectangle

P2D_Rect		
description	draw a rectangle	
argument	const rect_st *rec	rectangle window
return	none	

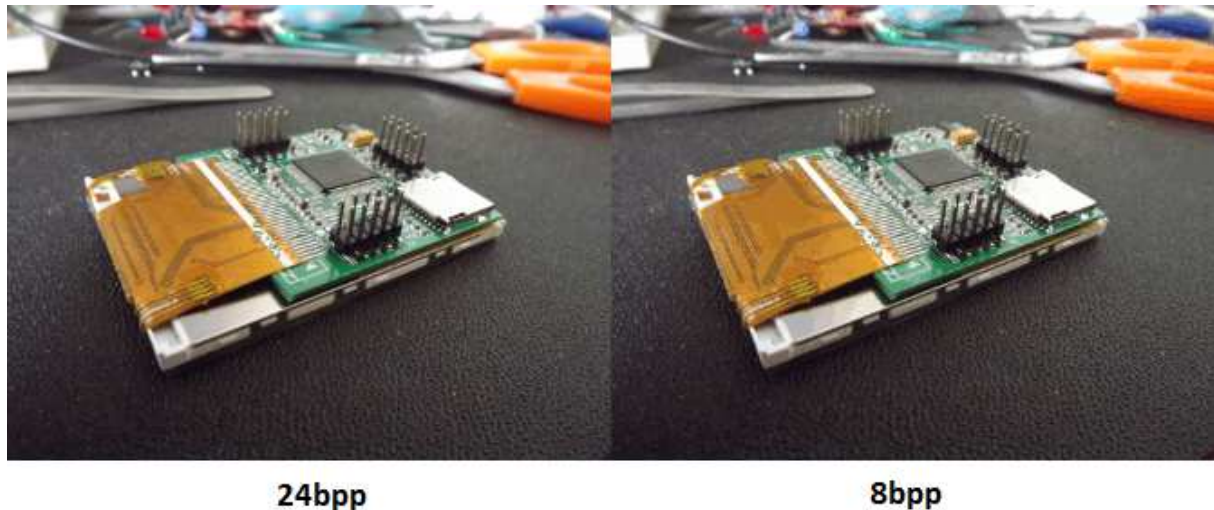
P2D_FillRect		
description	draw a filled rectangle	
argument	const rect_st *rec	rectangle window
return	none	
example	<pre>P2D_SetColor(COLOR_GREY); P2D_Clear(); rect_st rec = {10,10,100,20}; P2D_SetColor(COLOR_RED); P2D_Rect(&rec); P2D_SetLineType(LINE_DOT); rec.y += 30; P2D_Rect(&rec); rec.y += 30; P2D_FillRect(&rec);</pre>	

5.3. CLUTs

5.3.1. Introduction

CLUTs (Color Look-Up Tables) are used for saving memory without impacting performances (unlike *jpg* or *png* compression). It consists in reducing the number of colors composing an image (usually 2^{16} at least) into a color palette containing fewer colors (256 colors here; also known as [color quantization](#)). A 256 colors palette allows pixel to be coded with only one byte. By using optimal palette (provided by [gimp](#)), the 8bpp resulting image is quite identical to the 24bpp one, with much smaller memory footprint.

Figure 11: 24bpp vs 8bpp palletized



5.3.2. CLUT file description

CLUT file is very basic:

- The two first bytes compose a uint16, and represent the number of colors contained in the CLUT
- Each color entry is composed by 3 bytes, respectively red, green and blue channels.

Figure 12: CLUT file example

```
const uint8_t lutFile[] = {
    0x00, 0x10, /*Number of colors*/
    0xE4, 0xE7, 0xE5, Color #0
    0x82, 0xBA, 0x87, Color #1
    0xDD, 0xE1, 0xDE, ...
    0x59, 0x8F, 0x60,
    0x74, 0x9D, 0x7C,
    0x9B, 0xD0, 0xA0,
    0x39, 0x99, 0x40,
    0x01, 0x87, 0x00,
    0x3B, 0x94, 0x51,
    0x30, 0x7D, 0x36,
    0xA9, 0xCF, 0xAF,
    0x5E, 0xA4, 0x61,
    0x33, 0xB7, 0x63,
    0x10, 0x9E, 0x1D,
    0x8C, 0xB2, 0x91,
    0xBD, 0xD5, 0xD8, Color #15
};
```

Red Green Blue

5.3.3. CLUT creation

P2D_InitLut8BPP		
description	Initialize an 8BPP color LUT (256 different colors)	
argument	lut8bpp_st *lut	Pointer to an already allocated lut8bpp_st struct
	const uint8_t *pFile	Pointer to a 8BPP CLUT file; can be NULL, depending of mode
	lutmode_t mode	CLUT configuration flags; see §5.3.5
return	int8_t	0 if success, -1 otherwise
example	see §5.3.5	

P2D_InitLut4BPP		
description	Initialize a 4BPP color LUT (16 different colors)	
argument	lut4bpp_st *lut	Pointer to an already allocated lut4bpp_st struct
	const uint8_t *pFile	Pointer to a 4BPP CLUT file; can be NULL, depending of mode
	lutmode_t mode	CLUT configuration flags; see §5.3.5
return	int8_t	0 if success, -1 otherwise
example	see §5.3.5	

5.3.4. CLUT update

P2D_UpdateLut4BPP		
description	Update a 4BPP LUT: according to configuration flags and graphic context (e.g. <i>front & background color</i>), this function will re-compute each color composing the CLUT, only if needed.	
argument	lut4bpp_st *lut	Pointer to an already allocated lut4bpp_st struct
return	none	
example	see §5.3.5	

P2D_UpdateLut8BPP		
description	Update an 8BPP LUT: according to configuration flags and graphic context (e.g. <i>front & background color</i>), this function will re-compute each color composing the CLUT, only if needed.	
argument	lut8bpp_st *lut	Pointer to an already allocated lut8bpp_st struct
return	none	
example	see §5.3.5	

5.3.5. CLUT configuration flags

Table 10: CLUT flags

Symbol	Description
LUT_E_COPY	Copies a LUT file into a lut4bpp_st / lut8bpp_st struct. This operation is used for loading the CLUT of the sprites. User shall give the *pFile to P2D_InitLut4BPP / P2D_InitLut8BPP
LUT_E_FILLED	Fills the whole CLUT with the current <i>front color</i> .
LUT_E_GRADIENT	Fills the whole CLUT with a color gradient (from <i>front color</i> to <i>background color</i>). For example, this configuration is used for AA text.

LUT_O_ALPHA	Apply an alpha value to the whole CLUT; can be summarized as “For each color, $color = P2D_Alpha_a_on_b(color, background\ color, alpha)$ ”
LUT_O_BLACK_AND_WHITE	Convert all colors of a CLUT to black & white levels
LUT_O_COLOR_KEY	If defined, the first color of a CLUT will be equal to the <i>background color</i>

5.4. Text

5.4.1. Font file description

A font file is composed of the following parts:

- A header, containing the font type (1 for 1BPP, 2 for 4BPP), the ASCII code of the first and the last printable glyphs and the font height (in pixel; all glyphs have the same height)
- An entry table: each glyph has a width (in pixel) and a memory offset to its “displayable” content. This offset is relative to the displayable content start address
- The displayable content (or *raw*), located just after the offset table end. For each glyph, this content describes all pixels composing the glyph, from top-left to bottom-right:

0	0	0	0	3	F	F	F	F	3	0	0	0	0
0	0	0	0	6	F	F	F	F	6	0	0	0	0
0	0	0	0	C	F	F	F	F	D	0	0	0	0
0	0	0	0	5	F	F	E	F	F	4	0	0	0
0	0	0	0	8	F	F	6	8	F	8	0	0	0
0	0	0	1	F	F	F	4	5	F	F	1	0	0
0	0	0	5	F	F	C	0	1	C	F	5	0	0
0	0	0	9	F	F	5	0	0	9	F	9	0	0
0	0	3	F	F	F	3	0	0	5	F	F	3	0
0	0	5	F	F	A	0	0	0	1	F	F	6	0
0	0	C	F	F	F	F	F	F	F	F	E	0	0
0	4	F	F	F	F	F	F	F	F	F	F	3	0
0	8	F	F	9	0	0	0	0	0	E	F	8	0
1	F	F	F	5	0	0	0	0	0	6	F	F	2
5	F	F	F	3	0	0	0	0	0	4	F	F	5
9	F	F	9	0	0	0	0	0	0	C	F	F	A

```
uint8_t *glyph_a = {
    0x00, 0x00, 0x03, 0xFF, 0xFF, 0x30, 0x00, 0x00,
    0x00, 0x00, 0x06, 0xFF, 0xFF, 0x60, 0x00, 0x00,
    0x00, 0x00, 0x0C, 0xFF, 0xFF, 0xD0, 0x00, 0x00,
    0x00, 0x00, 0x5F, 0xFE, 0xFF, 0xF4, 0x00, 0x00,
    0x00, 0x00, 0x8F, 0xF6, 0x8F, 0xF8, 0x00, 0x00,
    0x00, 0x01, 0xFF, 0xF4, 0x5F, 0xFF, 0x10, 0x00,
    0x00, 0x05, 0xFF, 0xC0, 0x1C, 0xFF, 0x50, 0x00,
    0x00, 0x09, 0xFF, 0x50, 0x09, 0xFF, 0x90, 0x00,
    0x00, 0x3F, 0xFF, 0x30, 0x05, 0xFF, 0xF3, 0x00,
    0x00, 0x5F, 0xFA, 0x00, 0x01, 0xFF, 0xF6, 0x00,
    0x00, 0xCF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0x00,
    0x04, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x30,
    0x08, 0xFF, 0x90, 0x00, 0x00, 0x0E, 0xFF, 0x80,
    0x1F, 0xFF, 0x50, 0x00, 0x00, 0x06, 0xFF, 0xF2,
    0x5F, 0xFF, 0x30, 0x00, 0x00, 0x04, 0xFF, 0xF5,
    0x9F, 0xF9, 0x00, 0x00, 0x00, 0x00, 0xCF, 0xFA
};
```

Figure 13: Glyph raw example

For 1BPP glyphs, each pixel is stored in one bit. For 4BPP glyphs, each pixel is stored in 4 bits representing an *alpha* level (between 0x0 and 0xF, where 0x0 means transparent and 0xF means opaque). A random byte from the *displayable content* section belongs to one – and only one – glyph (e.g. if the 1BPP glyph ‘A’ use only 2 bits of its last byte, then the glyph ‘B’ will not use the 6 remaining bits; glyphs do not share unused bits).


```
const uint8_t FontFreeSerif_4bpp_12[] = {
    2, /*font type*/
    32, /*first car ID*/
    255, /*number of car*/
    14, /*font height*/
    /*index table: glyph width, offset hi, lh, ll*/
    0x03, 0x00, 0x00, 0x00,
    0x04, 0x00, 0x00, 0x10,
    0x05, 0x00, 0x00, 0x35,
    0x06, 0x00, 0x00, 0x57,
    0x06, 0x00, 0x00, 0x80,
    0x0b, 0x00, 0x00, 0xad,
    0x0a, 0x00, 0x00, 0xf8,
    0x03, 0x00, 0x01, 0x42,
    0x04, 0x00, 0x01, 0x55,
    0x05, 0x00, 0x01, 0x75,
    0x07, 0x00, 0x01, 0x99,
    0x07, 0x00, 0x01, 0xc8,
    0x04, 0x00, 0x01, 0xf8,
    0x04, 0x00, 0x02, 0x1a,
    0x03, 0x00, 0x02, 0x37,
    0x04, 0x00, 0x02, 0x4d,
    0x06, 0x00, 0x02, 0x6a,
    ...

    /*glyph raw*/
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x01, 0x40, 0x05, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x20, 0x00, 0x00, 0x00, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x90, 0x00, 0x70, 0xdc, 0xeb, 0x07, 0x42, 0x90, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x3f, 0x17, 0x1f, 0x3f, 0x01, 0x0b, 0x0f, 0x00, 0x01, 0xaf, 0xc1, 0x00, 0x2f, 0xc0, 0x10, 0x2f, 0x3f, 0x24, 0x2f, 0x3d,
    0x1a, 0x0f, 0x94, 0x00, 0x2f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x01, 0x9a, 0x00, 0x17, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x01, 0xf0, 0x07, 0x21, 0x00, 0x00, 0x06, 0x00, 0x03, 0x00, 0x00, 0x00, 0x04, 0x07, 0x62, 0x00, 0x01, 0x40,
    0xf1, 0x4d, 0x00, 0x00, 0x60, 0x1f, 0x09, 0x70, 0x00, 0x33, 0x00, 0xa8, 0x90, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x60, 0x01, 0xa7, 0xf3, 0x2d, 0x00, 0x0c, 0x30, 0xcc, 0x56, 0x00, 0x1f, 0x00, 0x3f, 0xe1, 0x00, 0x1f, 0x50, 0x1c, 0xf6,
    0x21, 0x07, 0xf8, 0x92, 0x8f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x1e, 0x00, 0xd0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x1f, 0x00, 0x00, 0x10, 0x00, 0x40, 0x01, 0x90, 0x00, 0x35, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x15, 0x00,
    0x00, 0x36, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3d, 0x00, 0x02, 0xf0, 0x00, 0x00, 0x00, 0x04, 0xc0, 0x00, 0x76,
    0x00, 0x0a, 0x00, 0x00, 0x20, 0x01, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x03, 0xf4, 0xc0, 0x01, 0xf8, 0x60, 0x01, 0x55, 0xf6, 0xb0, 0x01, 0x2f, 0x01, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x2f, 0x00, 0x0a, 0xee, 0xf2, 0x00, 0x2f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    ...

    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x5a, 0x00, 0x00, 0x3f, 0x00, 0x00, 0x2f, 0x00, 0x00, 0x2f, 0x7d, 0xd2,
    0x2f, 0x10, 0x90, 0x2f, 0x00, 0x3f, 0x2f, 0x00, 0x3e, 0x2f, 0x00, 0x68, 0x2f, 0x07, 0x00, 0x2f, 0x00, 0x00, 0x2f, 0x00,
    0x00, 0x7f, 0x50, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0xc0, 0x02, 0x10, 0x20,
    0x4f, 0x40, 0x77, 0xc0, 0x50, 0x71, 0x05, 0xc0, 0x00, 0x00, 0xe4, 0x50, 0x00, 0xef, 0x10, 0x00, 0x2a, 0x00, 0x00, 0x35,
    0x00, 0x23, 0x00, 0x00, 0x6f, 0x50, 0x00, 0x00
};
```

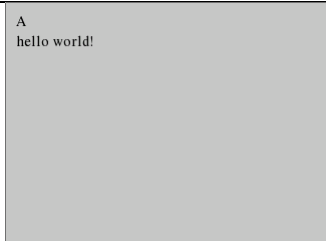
Figure 14: Font file example

5.4.2. Set a font

P2D_SetFont		
description	Set the font to use for next font-related operations	
argument	const uint8_t *pFile	Pointer to a font file. See resources.h header
return	int8_t	0 if success, -1 otherwise
example	<pre>if(P2D_SetFont(FontFreeSerif_4bpp_n_16) < 0) { //error } else { //success }</pre>	

5.4.3. Display some text

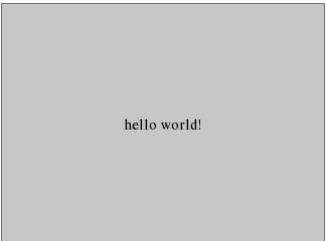
P2D_PutGlyph		
description	Display a glyph at a given position, by using the font previously set by P2D_SetFont	
argument	coord_t x, y	absolute coordinates of the top-left of the glyph
	uint8_t glyph	Glyph #id. If the glyph #id is not contained in the current font, nothing is displayed
return	none	

P2D_PutText		
description	Display a text at a given position, by using the font previously set by P2D_SetFont This functions does not handle ASCII controls (like '\n')	
argument	coord_t x, y	absolute coordinates of the top-left of the first glyph of the string
	const void *ptr	pointer to the string ('\0' terminated); will be internally casted in const uint8_t *
return	none	
example	<pre>P2D_SetColor(COLOR_LIGHT_GREY); P2D_Clear(); P2D_SetColors(COLOR_BLACK, COLOR_LIGHT_GREY); P2D_SetFont(FontFreeSerif_4bpp_n_16); P2D_PutGlyph(10, 10, 'A'); P2D_PutText(10, 30, "hello world!");</pre> 	

5.4.4. Get text dimension

P2D_GetTextHeight		
description	Return the height (in pixel) of the current font, previously set by P2D_SetFont	
argument	none	
return	length_t	font height, in pixel

P2D_GetGlyphwidth		
description	Return the width (in pixel) of a given glyph of the current font, previously set by P2D_SetFont	
argument	uint8_t g	the given glyph; if the glyph does not exist in the current font, the function will return 0
return	length_t	glyph width, in pixel

P2D_GetTextwidth		
description	Return the width (in pixel) of a '\0' terminated string, by using the font previously set by P2D_SetFont. The string width is the sum of all glyphs width.	
argument	const void *ptr	pointer to the string ('\0' terminated); will be internally casted in const uint8_t *
return	length_t	string width, in pixel
example	<pre>char *str = "hello world!"; coord_t x, y; P2D_SetColor(COLOR_LIGHT_GREY); P2D_Clear(); P2D_SetColors(COLOR_BLACK, COLOR_LIGHT_GREY); P2D_SetFont(FontFreeSerif_4bpp_n_16); x = LCD_GetWidth(); x -= P2D_GetTextwidth(str); x /= 2; y = LCD_GetHeight(); y -= P2D_GetTextHeight(); y /= 2; P2D_PutText(x, y, str);</pre> 	

5.5. Sprites



For now, only 8BPP sprites are implemented.

5.5.1. Sprite file description

Structure of a sprite file is quite similar to the font file one, and is composed of the following parts:

- A header, containing the sprite BPP (8 for 8BPP), the height and width of the sprite (in pixels, on 16 bits)
- The displayable content (or *raw*), located just after the header end. This content describes all pixels composing the sprite, from top-left to bottom-right:

Figure 15: Sprite file example

[illegible]

A sprite is always associated with a CLUT file; each pixel is defined by a color #id belonging to this CLUT.

5.5.2. Set a CLUT

P2D_SpriteSetLut8BPP		
description	Define the CLUT to use for displaying sprites.	
argument	const lut8bpp_st *lut	pointer to an already allocated CLUT
return	none	
example	See §5.5.4	


5.5.3. Get sprite dimension

P2D_SpriteGetHeight		
description	Return the height (in pixel) of a given sprite	
argument	const uint8_t *pFile	pointer to the sprite file
return	length_t	sprite height, in pixel; if the sprite does not exist, return 0

P2D_SpriteGetWidth		
description	Return the width (in pixel) of a given sprite	
argument	const uint8_t *pFile	pointer to the sprite file
return	length_t	sprite width, in pixel; if the sprite does not exist, return 0

5.5.4. Display a sprite

P2D_Sprite

description	Put a given sprite part on current surface, at given coordinates	
argument	const rect_st *src	selected part of the sprite to display; if NULL, select the whole sprite
	const rect_st *dst	destination; only dst->x & dst->y are used
	const uint8_t *pFile	pointer to the sprite file
return	none	
example	<pre> lut8bpp_st lut; rect_st dst, src; P2D_SetColors(COLOR_LIGHT_GREY, COLOR_LIGHT_GREY); P2D_Clear(); /*load the CLUT in RAM memory*/ P2D_InitLut8BPP(&lut, spriteLutFile03, LUT_E_COPY LUT_O_COLOR_KEY); /*put the whole sprite at x=y=10*/ dst.x = 10; dst.y = 10; P2D_SpriteSetLut8BPP(&lut); P2D_Sprite(NULL, &dst, spriteFile03); /*reload the CLUT in RAM memory, with alpha level*/ P2D_SetAlpha(64); P2D_InitLut8BPP(&lut, spriteLutFile03, LUT_E_COPY LUT_O_ALPHA LUT_O_COLOR_KEY); /*put the whole sprite at x=50 y=10*/ dst.x = 50; dst.y = 10; P2D_Sprite(NULL, &dst, spriteFile03); /*reload the CLUT in RAM memory, with B&W transformation*/ P2D_InitLut8BPP(&lut, spriteLutFile03, LUT_E_COPY LUT_O_BLACK_AND_WHITE LUT_O_COLOR_KEY); /*put the whole sprite at x=90 y=10*/ dst.x = 90; dst.y = 10; P2D_Sprite(NULL, &dst, spriteFile03); /*put the a part of the sprite (defined by src) at x=130 y=10*/ src.x = 10; src.y = 10; src.w = 10; src.h = 100; dst.x = 130; P2D_Sprite(&src, &dst, spriteFile03); </pre> 	

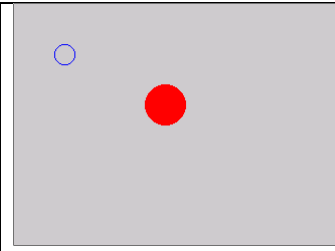
5.6. Geometrics

5.6.1. Draw a circle

P2D_Circle		
description	Draw a circle (1 pixel width)	
argument	coord_t x0, y0	center of the circle
	length_t radius	radius, in pixel
return	none	

P2D_FillCircle		
description	Draw a filled circle	
argument	coord_t x0, y0	center of the circle

	length_t radius	radius, in pixel
return	none	
example	<pre>P2D_SetColor(COLOR_BLUE); P2D_Circle(50, 50, 10); P2D_SetColor(COLOR_RED); P2D_FillCircle(150, 100, 20);</pre>	



5.6.2. Draw a polygon



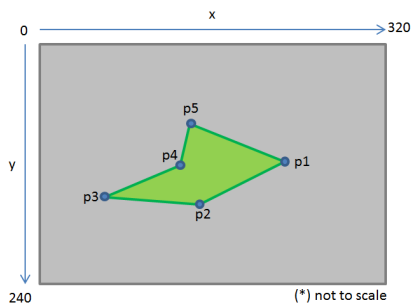
Implementation of filled polygon is inspired from http://alienryderflex.com/polygon_fill/

Table 11: polygon-related variables type

type	description
<pre>typedef struct { coord_t x, y; } point_st;</pre>	Point definition for polygon-related functions

5.6.2.1. Defining a polygon

It consists in enumerating all points composing the polygon, as shown in the example below:



```
#define NB_POINT 5
const point_st myPolygon[NB_POINT] = {
    {260, 120}, //p1
    {150, 170}, //p2
    {40, 165}, //p3
    {140, 125}, //p4
    {145, 70} //p5
};
```

5.6.2.2. Operation on points

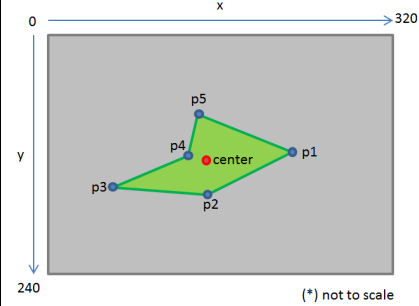


Successive modifications on the same array (like *zoom* or *rotate*) should be avoided due to accumulation of rounding errors.

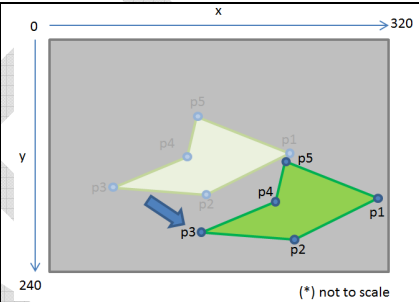
P2D_P_Copy		
description	Copy an array of points into another given array. Used for avoiding rounding issues.	
argument	const point_st *src	source array
	point_st *dst	destination array
	uint8_t nbPoint	number of points contained in the arrays
return	none	

P2D_FindPolyCenter		
description	Find the center of the polygon	
argument	const point_st *arPoints	polygon points
	uint8_t nbPoint	number of points contained in the arrays

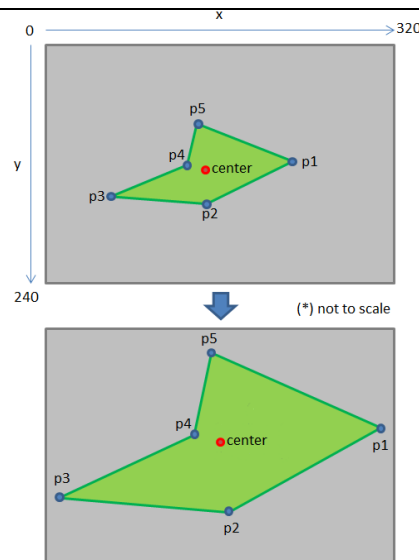
	point_st *center	returned center
return	none	
example	<pre>#define NB_POINT 5 point_st center; point_st myPolygon[NB_POINT] = { {260, 120}, {150, 170}, {40 , 165}, {140, 125}, {145, 70} }; P2D_FindPolyCenter(myPolygon, NB_POINT, &center);</pre>	



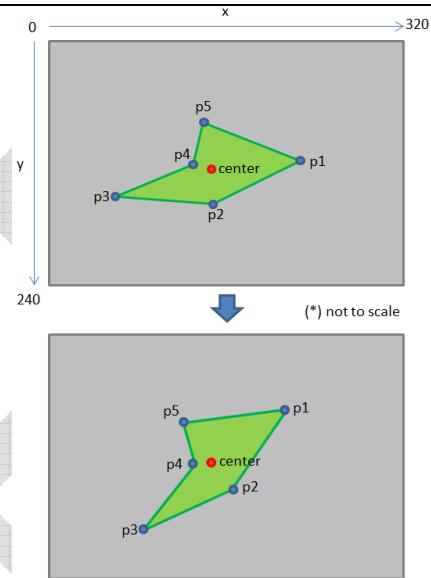
P2D_P_Move		
description	Move a polygon	
argument	point_st *arPoints	polygon points; will be overwritten
	uint8_t nbPoint	number of points contained in the arrays
	coord_t mvX, mvY	increment on x and y
return	none	
example	<pre>#define NB_POINT 5 point_st myPolygon[NB_POINT] = { {260, 120}, {150, 170}, {40 , 165}, {140, 125}, {145, 70} }; P2D_P_Move(myPolygon, NB_POINT, 100, 50);</pre>	



P2D_P_Zoom		
description	Zoom a polygon according to a reference point	
argument	point_st *arPoints	polygon points; will be overwritten
	uint8_t nbPoint	number of points contained in the arrays
	const point_st *ref	center of the zoom
	uint8_t zoomPercent	Zoom value, in percent (from 1% to 255%)
return	none	
example	<pre>#define NB_POINT 5 point_st center; point_st myPolygon[NB_POINT] = { {260, 120}, {150, 170}, {40 , 165}, {140, 125}, {145, 70} }; P2D_FindPolyCenter(myPolygon, NB_POINT, &center); P2D_P_Zoom(myPolygon, NB_POINT, &center, 200);</pre>	



P2D_P_Rotate		
description	Rotate a polygon according to a reference point	
argument	point_st *arPoints	polygon points; will be overwritten
	uint8_t nbPoint	number of points contained in the arrays
	const point_st *ref	center of the rotation
	uint16_t deg	rotation, in degrees
return	none	
example	<pre> #define NB_POINT 5 point_st center; point_st myPolygon[NB_POINT] = { {260, 120}, {150, 170}, {40 , 165}, {140, 125}, {145, 70} }; P2D_FindPolyCenter(myPolygon, NB_POINT, &center); P2D_P_Rotate(myPolygon, NB_POINT, &center, 45); </pre>	



5.6.2.3. Draw the polygon

P2D_Poly		
description	Draw a polygon	
argument	point_st *arPoints	polygon points
	uint8_t nbPoint	number of points of the polygon
return	none	

P2D_FillPoly		
description	Draw a filled polygon	
argument	point_st *arPoints	polygon points
	uint8_t nbPoint	number of points of the polygon
return	none	

5.6.2.4. Polygon example

Refer to <src/app/gui_demo/p2d_demo.c> file for a complete example.

5.6.3. Math functions

P2D_RandInit		
description	Initialize the pseudo random generator	
argument	uint32_t key	Seed value
return	none	

P2D_RandInit

description	Return a pseudo random number between 0 and a given maximum	
argument	uint16_t key	Maximum value
return	uint16_t	pseudo random value

P2D_Sin

description	Return sin(angle)	
argument	int16_t deg	angle (in degrees)
return	int16_t	sin(angle); between -32767 and +32767 (P2D_G_DIV define)
example	<pre>float s; s = (float) P2D_Sin(90) / P2D_G_DIV;</pre>	

P2D_Cos

description	Return cos(angle)	
argument	int16_t deg	angle (in degrees)
return	int16_t	cos(angle); between -32767 and +32767 (P2D_G_DIV define)
example	<pre>float s; s = (float) P2D_Cos(90) / P2D_G_DIV;</pre>	

P2D_Abs

description	Return abs(value)	
argument	int32_t val	input value
return	int32_t	val

P2D_sqrt

description	Return √(value)	
argument	uint16_t val	input value
return	uint8_t	√(value)

5.7. Utils

P2D_GetPixelCnt

description	Return the number of pixels contained in a given rect_st. Equivalent to rec->w * rec->h	
argument	const rect_st *rec	input rect_st
return	uint32_t	number of pixels contained in *rec

P2D_GetLengthBtwCoord

description	return the number of pixels between 2 coordinates belonging to the same type (e.g. x0 and x1 or y0 and y1, but not x0 and y0) Example: length between x0=10 and x1=13 is 4 (x0-x1 +1: there are 4 pixels between 10 & 13)	
argument	coord_t p0, p1	coordinates to compare
return	length_t	length

P2D_CoordGetMin

description	Return the lowest value between two coord_t	
-------------	---------------------------------------------	--

argument	coord_t p0, p1	coordinates to compare
return	coord_t	lowest coord_t

P2D_CoordGetMax

description	Return the highest value between two coord_t	
argument	coord_t p0, p1	coordinates to compare
return	coord_t	highest coord_t

P2D_CoordGetDelta

description	Return p0 - p1	
argument	coord_t p0, p1	input coordinates
return	coord_t	p0 - p1

P2D_IsInClip

description	Check if a point is into a given rectangle	
argument	const rect_st *rec	input rect_st
	coord_t p0, p1	input coordinates
return	bool	true if the point is in the rect_st, false otherwise

P2D_RectToCoord

description	Convert a rect_st into absolute coordinates	
argument	const rect_st *rec	input rect_st
	coord_t *x0, *y0, *x1, *y1	output coordinates
return	int8_t	0: success, -1: error (from NULL input pointer, or rec->w == 0 or rec->h == 0)

P2D_CoordToRect

description	Convert absolute coordinates into a rect_st	
argument	rect_st *rec	output rect_st
	coord_t x0, y0, x1, y1	input coordinates
return	int8_t	0: success, -1: error (from NULL input pointer)

5.8. Internal buffering



This module is no longer maintained and may cause corruptions due to the memory allocator used. Nevertheless, this feature is usable if no memory is allocated before the surface destruction (see <src/app/gui_demo/p2d_demo.c> file for example)

5.8.1. Create an internal surface

P2D_SurfaceCreate

description	Create an internal surface (located in RAM memory)	
argument	rect_st *rec	dimension of the surface (only w & h are used)

return	surfaceId_t	surface #id if success (range[SURFACE_1-(SURFACE_NUMBER-1)], SURFACE_LCD if error (e.g. not enough memory)
--------	-------------	------------------------------------------------------------------------------------------------------------

5.8.2. Set the destination

Destination switching (drawing on the screen or on a memory buffer) is made by redirecting the 5 basic interfaces (See §5.1.3). If the destination is hardware defined (i.e. the TFT screen), interfaces are linked to the ILI9320 driver; if the destination is software defined (i.e. internal buffered), interfaces are linked to local functions (in p2d_buffer.c file)

P2D_SetDest		
description	Set the current surface; once set, all P2D operations will be performed on this surface	
argument	surfaceId_t	desired surface destination
return	surfaceId_t	identical to the desired destination if success, SURFACE_LCD if error

5.8.3. Copy a surface to screen

P2D_CopySurface		
description	Copy a part of a surface to the current one, at specified coordinates	
argument	surfaceId_t	source surface (shall be != of the current one and != of SURFACE_LCD)
	const rect_st *src	dimension of the part of the source to copy; copy the whole source if NULL
	const rect_st *dst	destination coordinates (only x & y are used)
return	none	

5.8.4. Delete an internal surface

P2D_SurfaceDelete		
description	delete an internal surface and <u>free the allocated memory</u> ⁽¹⁾	
argument	surfaceId_t	source surface (shall be != of the current one and != of SURFACE_LCD)
return	none	

⁽¹⁾ The function calls `sfreefrom()`, which frees memory from the surface start address to the end of allocable area. This implies that nothing shall be allocated before deleting the surface; otherwise, when deleting the surface, next allocated objects will be erased or corrupted.

5.8.5. Example

Refer to `<src/app/gui_demo/p2d_demo.c>` file, where the flickering effect is avoided thanks to internal buffering usage.

6. GUI

6.1. Overview

GUI (stands for *Graphical User Interface*) is a set of functions which perform object-oriented graphical operations (e.g. draw & handle buttons, lists, sliders...).

GUI is written in C language only.

6.1.1. Generic object



This part is internally handled by the GUI, and is described only for documentation purpose. User should not directly access / modify such data

Each widget is derived from a GUI generic object (g_obj_st structure) that contains all common parameters, like position & dimension on screen, object group, attributes... Structure of generic object is defined in <app/gui/gui_obj.h> at line 52:

```

48  /**
49   * @struct: g_obj_st
50   * @brief: generic object definition
51   */
52  typedef struct g_obj_t {
53
54      struct g_obj_t /*@null@*/ *next; /*next generic object*/
55
56      void *obj; /*physical object*/
57      pObjFunc_t /*@null@*/ task; /*task of the object; this one handles specific object refresh*/
58      pObjFunc_t /*@null@*/ draw; /*draw function of the object*/
59
60      rect_st rec; /*absolute object window*/
61      signal_t signals[E_MAX_EVENT]; /*signals table (one signal per possible event)*/
62      state_t state; /*object state (disabled, hidden, focused, etc...)*/
63      event_e event; /*user event (released, pushed, ...)*/
64      group_t group; /*object group*/
65
66  } g_obj_st;

```

6.1.2. Object attributes

Each generic object has attributes, handled by the GUI itself. These attributes are listed above:

Table 12: generic object attribute

CLUT symbol	Description
NOTIFIED	Set by the user; if true, the GUI shall redraw the object every 500ms. The notification drawing is handled by the widget itself, and is not available for all widgets (e.g. a button is notifiable, a text area is not).
BLINK	Dynamically set by the GUI; toggles every 500ms, used for notification drawing
PRESSED	Dynamically set by the GUI; true if touchscreen area corresponding to the object is pressed.
DISABLED	Set by the user; if true, the GUI shall redraw the object by specifying to the widget the disable state (e.g. a disabled button will be displayed in gray levels, and will be stuck to <i>released</i> position)
FOCUSABLE	Set by the widget

FOCUSED	Dynamically set by the GUI; an object is focused if it is both <i>focusable</i> and <i>pressed</i> . (e.g. a focused button will be displayed with a dashed rectangle)
NEED_REFRESH	May be set by the GUI, the widget or the user; if true, the GUI shall redraw the object.
STATIC	Set by the widget; if true, GUI will not redraw the item if this one is <i>pressed</i> (e.g. a text area does not need to be redrawn if <i>pressed</i> , unlike a button)

GUI_ObjIsNotified

description	return the NOTIFIED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
return	bool: true if notified, false otherwise	

GUI_ObjSetNotified

description	set the NOTIFIED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
	bool p	new flag value
return	none	

GUI_ObjIsPressed

description	return the PRESSED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
return	bool: true if pressed, false otherwise	

GUI_ObjSetPressed

description	set the PRESSED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
	bool p	new flag value
return	none	

GUI_ObjIsDisabled

description	return the DISABLED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
return	bool: true if disabled, false otherwise	

GUI_ObjSetDisabled

description	set the DISABLED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
	bool p	new flag value
return	none	

GUI_ObjIsFocusable

description	return the FOCUSABLE flag of a generic object	
-------------	-----------------------------------------------	--

argument	const g_obj_st *obj	pointer to the generic object
return	bool: true if focusable, false otherwise	

GUI_ObjSetFocusable

description	set the FOCUSABLE flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
	bool p	new flag value
return	none	

GUI_ObjIsFocused

description	return the FOCUSED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
return	bool: true if focused, false otherwise	

GUI_ObjSetFocused

description	set the FOCUSED flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
	bool p	new flag value
return	none	

GUI_ObjIsNeedRefresh

description	return the NEED_REFRESH flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
return	bool: true if need to refresh, false otherwise	

GUI_ObjSetNeedRefresh

description	set the NEED_REFRESH flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
	bool p	new flag value
return	none	

GUI_ObjIsStatic

description	return the STATIC flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
return	bool: true if static, false otherwise	

GUI_ObjSetStatic

description	set the STATIC flag of a generic object	
argument	const g_obj_st *obj	pointer to the generic object
	bool p	new flag value
return	none	

6.1.3. Group

Generic objects can be grouped according to an id; this allows applying common attributes to several objects at the same time.

GUI_SetGroup		
description	Set the current group value; this current group value will be used for the next created objects	
argument	group_t g	new current group value
return	none	

GUI_GroupDisable		
description	Disable or enable all objects belonging to a given group	
argument	group_t g	group to enable / disable
	bool p	new flag value
return	none	

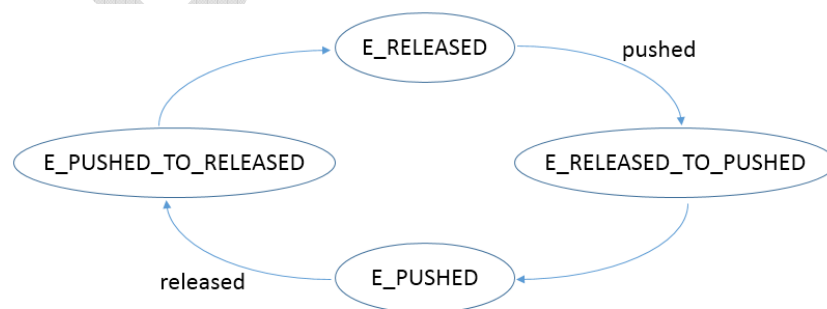
GUI_GroupNotify		
description	Notify or not all objects belonging to a given group	
argument	group_t g	group to notify
	bool p	new flag value
return	none	

GUI_GroupRefresh		
description	Refresh all objects belonging to a given group	
argument	group_t g	group to refresh
	bool p	new flag value
return	none	

6.1.4. Signals

For each object, GUI handles a small FSM composed of the following states:

- E_RELEASED: the object is released
- E_RELEASED_TO_PUSHED: the user has touched the object; this state lasts only one software cycle
- E_PUSHED: the object stays in this state while the user presses it.
- E_PUSHED_TO_RELEASED: the user has just released the object; this state lasts only one software cycle before going to E_RELEASED



* One transition (max) per software cycle

Figure 16: event FSM (simplified)

For each object, GUI allows to define a signal for each possible state through the following function:

GUI_SetSignal		
description	associate a signal to an event for the last added object	
argument	event_e event	Event (see Figure 16: event FSM)
	signal_t signal	Signal value (0 means no signal; useful signal from 1 to 0xFFFF)
return	none	

See §6.1.5 for signal catching.

6.1.5. Page creation / page handling / page jump

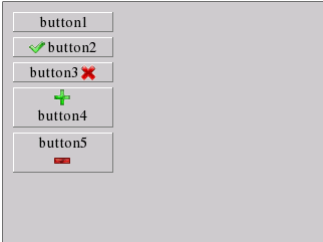
todo

6.2. Widgets


6.2.1. Button

GUI_W_ButtonAdd		
description	Add a basic button	
argument	const rect_st *rec	button dimension & position
	const void *str	String of the button; can be NULL
	gui_img_t img	Image of the button; can be == 0 (no image)
return	g_obj_st	

GUI_W_ButtonSetPlacementOrder		
description	Define the placement of the text and the image of buttons	
argument	btn_placeOrder_e order	<p>G_BTN_PLACE_IMG_L_TEXT_R: image and text on the same y; image at left, text at right</p> <p>G_BTN_PLACE_IMG_R_TEXT_L: image and text on the same y; image at right, text at left</p> <p>G_BTN_PLACE_IMG_T_TEXT_B: image and text on the same x; image at top, text at bottom</p> <p>G_BTN_PLACE_IMG_B_TEXT_T: image and text on the same x; image at bottom, text at top</p>
return	none	
Example	<pre>rect_st rec = {10, 10, 100, 20}; GUI_W_ButtonAdd(&rec, "button1", G_IMG_NONE); rec.y += rec.h + 5; GUI_W_ButtonAdd(&rec, "button2", G_IMG_OK); rec.y += rec.h + 5; GUI_W_ButtonSetPlacementOrder(G_BTN_PLACE_IMG_R_TEXT_L); GUI_W_ButtonAdd(&rec, "button3", G_IMG_NO); rec.y += rec.h + 5; rec.h = 40; GUI_W_ButtonSetPlacementOrder(G_BTN_PLACE_IMG_T_TEXT_B); GUI_W_ButtonAdd(&rec, "button4", G_IMG_ADD); rec.y += rec.h + 5;</pre>	

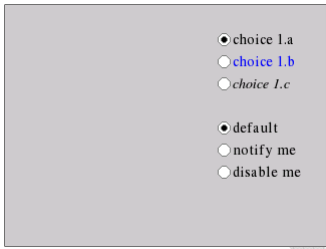
	<pre>GUI_W_ButtonSetPlacementOrder(G_BTN_PLACE_IMG_B_TEXT_T); GUI_W_ButtonAdd(&rec, "button5", G_IMG_REMOVE);</pre> 
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.2.2. Checkbox

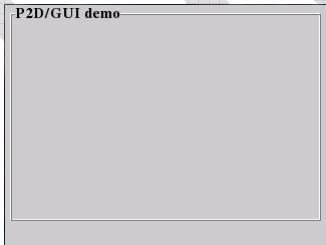
GUI_W_CheckBoxAdd		
description	Add a basic checkbox	
argument	const rect_st *rec	checkbox dimension & position
	const void *str	String of the checkbox; can be NULL
	const uint8_t *pState	Pointer to the state of the checkbox; if *pState == 0, the checkbox is not ticked. The widget does not directly modify *pState; the user shall handle *pState manually
return	g_obj_st	Pointer to the generic object if success, NULL if error
Example	<pre>uint8_t check1 = check2 = check3 = check4 = check5 = 0; rec = GUI_Rect(110, 25, 90, 19); GUI_W_CheckBoxAdd(&rec, "normal", &check1); rec = GUI_Rect(110, 47, 90, 19); SetColor(G_COL_TEXT, P2D_Color(0, 140, 0)); GUI_W_CheckBoxAdd(&rec, "color", &check2); SetColor(G_COL_TEXT, P2D_Color(0, 0, 0)); rec = GUI_Rect(110, 69, 90, 19); SetFont(G_FONT_ITALIC); GUI_W_CheckBoxAdd(&rec, "italic", &check3); SetFont(G_FONT_DEFAULT); rec = GUI_Rect(110, 91, 90, 19); GUI_W_CheckBoxAdd(&rec, "notify me", &check4); rec = GUI_Rect(110, 113, 90, 19); GUI_W_CheckBoxAdd(&rec, "disable me", &check5);</pre> 	

6.2.3. Radio

GUI_W_RadioAdd		
description	Add a basic radio	
argument	const rect_st *rec	radio dimension & position
	const void *str	String of the radio; can be NULL
	const uint8_t *pCurrentId	Pointer to the value of the #id of the selected radio; if *pCurrentId is equal to the radio's #id, then the radio is ticked. The widget does not directly modify *pCurrentId; the user shall handle *pCurrentId manually

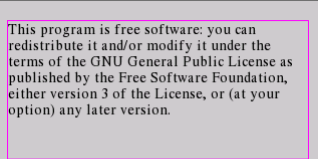
	uint8_t id	The #id of the radio
return	g_obj_st	Pointer to the generic object if success, NULL if error
Example	<pre> rec = GUI_Rect(210, 25, 90, 19); GUI_W_RadioAdd(&rec, "choice 1.a", &radio1, 1); rec = GUI_Rect(210, 47, 90, 19); SetColor(G_COL_TEXT, P2D_Color(0, 0, 255)); GUI_W_RadioAdd(&rec, "choice 1.b", &radio1, 2); SetColor(G_COL_TEXT, P2D_Color(0, 0, 0)); rec = GUI_Rect(210, 69, 90, 19); SetFont(G_FONT_ITALIC); GUI_W_RadioAdd(&rec, "choice 1.c", &radio1, 3); SetFont(G_FONT_DEFAULT); rec = GUI_Rect(210, 113, 90, 19); GUI_W_RadioAdd(&rec, "default", &radio2, 1); rec = GUI_Rect(210, 135, 90, 19); GUI_W_RadioAdd(&rec, "notify me", &radio2, 2); rec = GUI_Rect(210, 157, 90, 19); GUI_W_RadioAdd(&rec, "disable me", &radio2, 3); </pre> 	

6.2.4. Frame

GUI_W_FrameAdd		
description	Add a text frame	
Argument	const rect_st *rec	frame dimension & position
	const void *str	String of the frame; can be NULL
return	g_obj_st	Pointer to the generic object if success, NULL if error
Example	<pre> rec = GUI_Rect(5, 0, 310, 215); SetFont(G_FONT_BOLD); GUI_W_FrameAdd(&rec, "P2D/GUI demo"); </pre> 	

6.2.5. Text area

GUI_W_TextAdd		
description	Add a text area	
Argument	const rect_st *rec	area dimension & position
	const void *str	String of the area; can be NULL
return	g_obj_st	Pointer to the generic object if success, NULL if error
Example	<pre> rec = GUI_Rect(10, 20, 299, 137); GUI_W_TextAdd(&rec, "This program is free software: you can redistribute it and/or modify" </pre>	

	<pre>"it under the terms of the GNU General Public License as published by" "the Free Software Foundation, either version 3 of the License, or " "(at your option) any later version."); rec.x--; rec.y--; rec.w += 2; rec.h += 2; P2D_SetLineType(LINE_SOLID); P2D_SetColor(COLOR_PURPLE); P2D_Rect(&rec);</pre> 
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.2.6. Graph

GUI_W_GraphSetGridSpacing		
description	set the grid spacing	
Argument	length_t h	horizontal spacing, in pixel (min = 2px)
	length_t v	vertical spacing, in pixel (min = 2px)
return	none	

GUI_W_GraphAdd		
description	add graph container	
Argument	const rect_st *rec	graph dimension & position
	e_grid_type grid	Grid type: GRAPH_GRID_DISABLED: no grid GRAPH_GRID_DOT_H: horizontal grid GRAPH_GRID_DOT_V: vertical grid GRAPH_GRID_DOT_HV: horizontal & vertical grid
	uint16_t refreshTime	refresh period, in ms
return	g_obj_st*	pointer to the associated generic object if succeeded, NULL if error

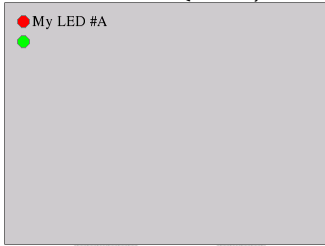
GUI_W_GraphAddCurveToGraph		
description	add a curve into a graph container; number of curve not limited	
Argument	g_obj_st *obj	pointer to the graph container
	uint8_t *data	pointer to the curve data (array of w uint8_t, where w is the width of the graph)
	color_t color	color of the curve
return	int8_t	0 ok, -1 error

GUI_W_GraphAddSampleToCurve		
description	add a sample to the curve; by using this function, corresponding curve will be considered as a circular buffer	
Argument	g_obj_st *obj	pointer to the graph container

	uint8_t curveId	curve ID (the first curve has ID = 0)
	uint8_t sample	new sample
return	none	

Refer to <src/app/gui_demo/gui_demo_graph1.c> and <src/app/gui_demo/gui_demo_graph2.c> for examples.

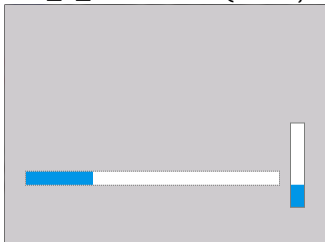
6.2.7. Led

GUI_W_LedAdd		
description	Add a text area	
Argument	const rect_st *rec	Led dimension & position
	const void *str	String of the led; can be NULL
	const uint8_t *pState	pointer to the state of the led (red if *pstate == 0, green otherwise)
return	g_obj_st	Pointer to the generic object if success, NULL if error
Example	<pre>uint8_t a = 0, b = 1; rect_st rec = GUI_Rect(10, 10, 100, 17); GUI_W_LedAdd(&rec, "My LED #A", &a); rec.y += 20; GUI_W_LedAdd(&rec, NULL, &b);</pre> 	

6.2.8. List

See §6.3.3

6.2.9. Slider

GUI_W_SliderAdd		
description	Add a slider	
Argument	const rect_st *rec	area dimension & position
	uint8_t *pvalue	Pointer to the slider value (in & out)
return	g_obj_st	Pointer to the generic object if success, NULL if error
Example	<pre>rect_st rec; uint8_t u8Slider = 64; rec = GUI_Rect(20, 165, 255, 15); GUI_W_SliderAdd(&rec, &u8Slider); rec = GUI_Rect(285, 117, 15, 85); GUI_W_SliderAdd(&rec, &u8Slider);</pre> 	

6.2.10. Value box

todo

6.2.11. Dial

todo

6.2.12. Value box (special Dial)

todo

6.2.13. User entry

todo

6.3. Macros

todo

6.3.1. Keyboard

todo

6.3.2. Popup

todo

6.3.3. Lists (macro)

todo

6.3.4. Files browser

todo

7. SD card



The smart TFT includes the FatFS from http://elm-chan.org/fsw/ff/00index_e.html which has been adapted to the PIC32 by following thread <http://www.microchip.com/forums/m563218.aspx>



The μ SD card is mounted once, at startup. File system will not be available if the μ SD card is plugged later, at runtime.

Refer to the official FatFS documentation available at http://elm-chan.org/fsw/ff/00index_e.html

8. RAM memory organization



This part is internally handled by the smart TFT, and is described only for documentation purpose. User should not modify memory management

The RAM memory is split in 4 areas:

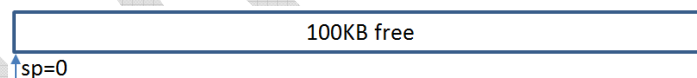
- *Allocable*: this contiguous area is used by the custom memory allocator, for dynamically providing memory to the GUI (default size = 100KB)
- *SSM*: shared memory between the smart TFT and a remote master (default size = 5KB)
- *Data*: few KB, composed of scattered small areas, defined by the compiler itself for storing program data
- *Unused*: few remaining KB, not used, available for user.

This chapter deals with the *allocable* and *SSM* memory areas.

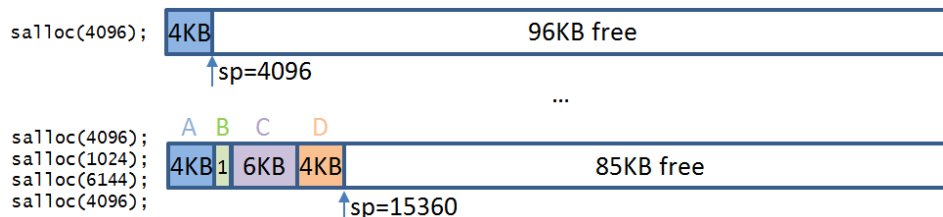
8.1. Memory allocator

The smart TFT uses a custom memory allocator (*stack allocator* like) which acts as follow:

- 1) A contiguous bounded area is defined at compilation time (here 100KB). A *stack pointer* is used to store the next allocable address



- 2) When a widget asks for memory (through *salloc()* function), the allocator shares a part of its area and updates its *stack pointer*. After a memory allocation, the *stack pointer* will always be aligned to the upper address aligned @4, if not already aligned.



- 3) Unlike the standard *free()* function which allows to release any allocated area (e.g. release only the **B** area), the custom allocator can only release from an allocated area address (e.g. releasing **B** will also release **C** and **D**), through the *sfreefrom()* function.



Notice that the *sfreefrom()* function will only update the stack pointer; content of **B**, **C** & **D** will not be modified until the next allocation.

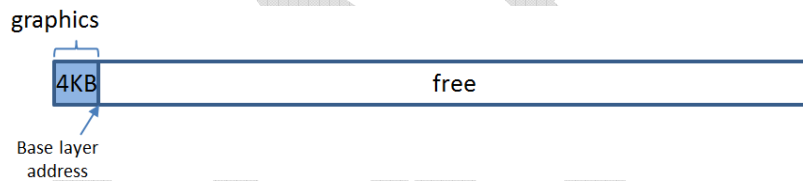
This strange releasing policy may be considered as a drawback, but has actually some advantages:

- It is simple to implement, and is very fast to execute (equivalent to $O(1)$ complexity)
- Since the allocation is contained in a bounded area, and since memory releasing is from a defined address (handled by the GUI, when creating a new page), memory leaks are not possible.
- Implementation of widgets destructors is useless

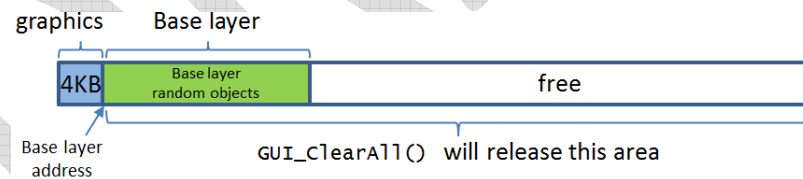
8.2. Allocable memory organization

At runtime, allocable memory could be split in 3 parts:

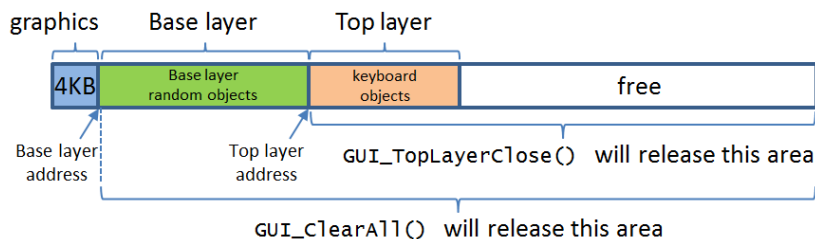
- **Graphics:** at startup, the GUI registers graphical resources to make their exploitation easier (e.g. a sprite will be handled with a simple #id instead of a pointer to a sprite file + a pointer to a CLUT file). Output of this registration is stored once, at the very beginning of allocable memory.



- **Base layer objects:** once the graphic registration completed, the GUI save the current stack pointer of the allocator. This pointer will remain constant and will be referenced as the address to use with *sfreefrom()* when clearing a page.



- **Top layer objects:** some macros (e.g. the built-in keyboard) use the top layer. When the top layer is opened, the GUI saves the current stack pointer of the allocator. This pointer will be used with *sfreefrom()* when closing the top layer.



8.3. Serialized Shared Memory



This part is effective only if the operational mode is defined to *slave* (see §3.7)

Some widgets need locally-referenced variables to work (e.g. a checkbox shall be linked to a `uint8_t`, and will display – or not – the tick according to this `uint8_t` value). The Serialized Shared Memory module is a small software part providing reading, writing and referencing services through a serial communication port.

The SSM module handle a small memory area (default size = 2KB) which is shared between a master (through UART) and the smart TFT itself. This memory area is accessed by offsets instead of addresses, where offset #0 is the start address of the area, offset #1 is the start address + 1 ...

The following functions are provided:

SSM_writeU8		
description	Write an <code>uint8_t</code> at a given offset	
Argument	<code>uint16_t</code> offset	offset, @1 aligned
	<code>uint8_t</code> val	new value to write
return	<code>int8_t</code>	0 success, -1 error (e.g. invalid offset)

SSM_writeU16		
description	Write an <code>uint16_t</code> at a given offset	
Argument	<code>uint16_t</code> offset	offset, @2 aligned
	<code>uint16_t</code> val	new value to write
return	<code>int8_t</code>	0 success, -1 error (e.g. invalid offset)

SSM_writeI16		
description	Write an <code>int16_t</code> at a given offset	
Argument	<code>uint16_t</code> offset	offset, @2 aligned
	<code>int16_t</code> val	new value to write
return	<code>int8_t</code>	0 success, -1 error (e.g. invalid offset)

SSM_writeU32		
description	Write an <code>uint32_t</code> at a given offset	
Argument	<code>uint16_t</code> offset	offset, @4 aligned
	<code>uint32_t</code> val	new value to write
return	<code>int8_t</code>	0 success, -1 error (e.g. invalid offset)

SSM_writeI32		
description	Write an <code>int32_t</code> at a given offset	
Argument	<code>uint16_t</code> offset	offset, @4 aligned
	<code>int32_t</code> val	new value to write
return	<code>int8_t</code>	0 success, -1 error (e.g. invalid offset)

SSM_ReadU8		
description	Read an <code>uint8_t</code> at a given offset	
Argument	<code>uint16_t</code> offset	offset, @1 aligned
	<code>uint8_t</code> *val	read result
return	<code>int8_t</code>	0 success, -1 error (e.g. invalid offset)

SSM_ReadU16		
description	Read an uint16_t at a given offset	
Argument	uint16_t offset	offset, @2 aligned
	uint16_t *val	read result
return	int8_t	0 success, -1 error (e.g. invalid offset)

SSM_ReadI16		
description	Read an int16_t at a given offset	
Argument	uint16_t offset	offset, @2 aligned
	int16_t *val	read result
return	int8_t	0 success, -1 error (e.g. invalid offset)

SSM_ReadU32		
description	Read an uint32_t at a given offset	
Argument	uint16_t offset	offset, @4 aligned
	uint32_t *val	read result
return	int8_t	0 success, -1 error (e.g. invalid offset)

SSM_ReadI32		
description	Read an int32_t at a given offset	
Argument	uint16_t offset	offset, @4 aligned
	int32_t *val	read result
return	int8_t	0 success, -1 error (e.g. invalid offset)

The example below shows the implementation of a simple radio, both at TFT side and master side:

TFT side (embedded app mode)	Master side (slave mode)
<pre> #define SIG_CHK 1 uint8_t chk1 = 0; void MyPageCreate(void) { rect_st rec; rec = GUI_Rect(210, 25, 90, 19); GUI_W_CheckBoxAdd(&rec, "check", &chk1); GUI_SetSignal(E_PUSHED_TO_RELEASED, SIG_CHK); ... } void MyPageHandle(signal_t sig) { if(sig == SIG_CHK) chk1 = !chk1; } </pre>	<pre> #define SIG_CHK 1 #define ADDR_CHK1 0 uint8_t chk1 = 0; void MyPageCreate(void) { rect_st rec; rec = GUI_Rect(210, 25, 90, 19); R_GUI_W_CheckBoxAdd(&rec, "check", ADDR_CHK1); R_MEM_WriteU8(ADDR_CHK1, chk1); GUI_SetSignal(E_PUSHED_TO_RELEASED, SIG_CHK); ... } void MyPageHandle(signal_t sig) { if(sig == SIG_CHK) { chk1 = !chk1; R_MEM_WriteU8(ADDR_CHK1, chk1); } } </pre>

At TFT side, the radio is just linked to its variable; at master side, the radio is linked to a uint8_t located in the SSM area at offset #0.

9. UART driving



This part is still under development



Limitation: for now, UART driving supports only the PIC32 endianness. This implies that the system used for driving the smart TFT shall send LITTLE_ENDIAN data.



Communication between the smart TFT and a master is half-duplex oriented; the master shall always wait for an answer before sending a new message.



An UART driving example (master side) is given in folder <remote_example>

The following chart will be applied for this whole paragraph:

Box color	Description
	indicates an unsigned char (uint8_t) parameter
	indicates a signed char (int8_t) parameter
	indicates an unsigned short (uint16_t) parameter
	indicates a signed short (int16_t) parameter
	indicates an unsigned integer (uint32_t) parameter
	indicates a signed integer (int32_t) parameter
	string ('\0' terminated) parameter

9.1. Message

A message (both from master to TFT or TFT to master) is constituted of 3 parts:

- *Start sequence*, defined by two successive TOKEN_START (x55) plus a sequence number (seq)
- *User message*, variable sized
- *Stop sequence*, defined by two successive TOKEN_STOP (xAA)

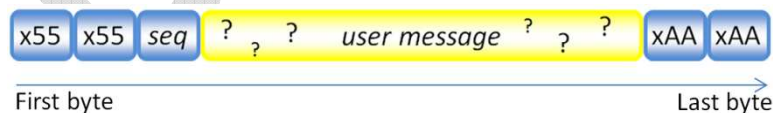


Figure 17: message structure

To avoid untimely *stop sequences* in the *user message*, a TOKEN_SKIP (xFE) is automatically inserted when the *user message* contains two successive xAA bytes. The TOKEN_SKIP acts the same way than the backslash '\ ' in a C file:

useful content to send	actual sent content
x01 xAA xAA x02	x01 xAA xFE xAA x02
x01 xFE x02	x01 xFE xFE x02

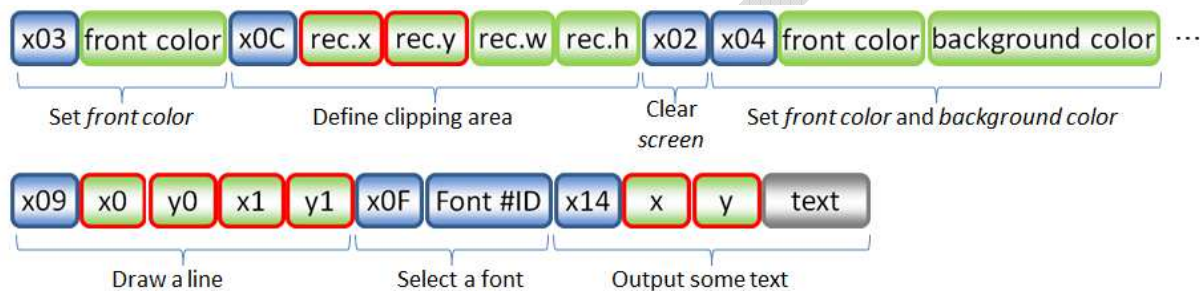
The sequence number is just a robustness feature handled at master side. For each received message, the smart TFT just *echoes* the sequence number to the master, ensuring that the processed message is the expected one. The sequence number can be stuck by the master to a constant value if not used.

9.2. User message (from master to smart TFT)

The *user message* contains a variable-sized list of variable-sized commands: a message can contain just one operation (e.g. clear the screen – 1 byte), or several operations (e.g. clear the screen – 1 byte, set red color – 3 bytes, and draw a line – 9 bytes).

Available commands are listed in §Serialized commands9.4.

A user message (from master to smart TFT) example is given below:



9.3. Answer message (from smart TFT to master)

Each time a message is received, the smart TFT processes it and sends back an answer message to the master. This answer message has the same form than described in §9.1. According to commands previously received, the answer size may vary; e.g.:

received message	answer content
clear the screen set RED color draw a line	0x00 (1 byte) ↔ success
write a byte located at offset = invalid	0xFF (1 byte) ↔ error
read the byte located at offset = 10 get screen width read the string located at offset = 200	byte at offset = 10 (1 byte), screen width (2 bytes), string at offset = 200 (n bytes), 0x00 (success, 1 byte)

9.4. Serialized commands

Each command is described as follow:

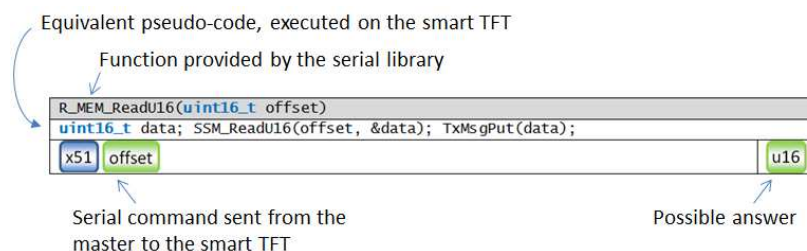


Figure 18: command description

reserved					
reserved					
x00					--
R_GUI_Debug(bool bDisplay)					
GUI_DBG_DispMemUsage(bDisplay);					
x01	bDisplay				--
R_P2D_Clear(void)					
P2D_Clear();					
x02					--
R_P2D_SetColor(color_t front_color)					
P2D_SetColor(front_color);					
x03	front color				--
R_P2D_SetColors(color_t front_color, color_t background_color)					
P2D_SetColors(front_color, background_color);					
x04	front color	background color			--
R_P2D_SetDisplayMode(dmode_t mode)					
P2D_SetDisplayMode(mode);					
x05	mode				--
R_P2D_P2D_SetAlpha (uint8_t alpha)					
P2D_SetAlpha(alpha);					
x06	alpha				--
R_P2D_SetLineType(line_t type)					
P2D_SetLineType(type);					
x07	type				--
R_P2D_SetPixel(coord_t x, coord_t y, color_t color)					
P2D_SetPixel(x, y, color);					
x08	x	y	color		--
R_P2D_Line(coord_t x0, coord_t y0, coord_t x1, coord_t y1)					
P2D_Line(x0, y0, x1, y1);					
x09	x0	y0	x1	y1	--
R_P2D_Rect(rect_st *rec)					
P2D_Rect(&rec);					
x0A	rec.x	rec.y	rec.w	rec.h	--
R_P2D_FillRect(rect_st *rec)					
P2D_FillRect(&rec);					
x0B	rec.x	rec.y	rec.w	rec.h	--

R_P2D_SetClip(rect_st *rec)	
P2D_SetClip(&rec);	
x0C rec.x rec.y rec.w rec.h	--

R_P2D_GetLcdwidth(void)	
w = P2D_GetLcdwidth(); TxMsgPut(w);	
x0D	width

R_P2D_GetLcdHeight(void)	
h = P2D_GetLcdHeight(); TxMsgPut(h);	
x0E	height

R_P2D_SetFont(gui_font_t font)	
SetFont(font_id);	
x0F Font #ID	--

R_P2D_GetTextHeight(void)	
h = P2D_GetTextHeight(); TxMsgPut(h);	
x10	height

R_P2D_GetTextWidth(void *str)	
w = P2D_GetTextWidth(text); TxMsgPut(w);	
x11 text	width

R_P2D_GetGlyphwidth(uint8_t glyph)	
w = P2D_GetGlyphwidth(glyph); TxMsgPut(w);	
x12 glyph	width

R_P2D_PutGlyph(coord_t x, coord_t y, uint8_t glyph)	
P2D_PutGlyph(x, y, glyph);	
x13 x y glyph	--

R_P2D_PutText(coord_t x, coord_t y, void *str)	
P2D_PutText(x, y, text);	
x14 x y text	--

R_P2D_Circle(coord_t x, coord_t y, length_t radius)	
P2D_Circle(x, y, radius);	
x15 x y radius	--

R_P2D_FillCircle(coord_t x, coord_t y, length_t radius)	
P2D_FillCircle(x, y, radius);	
x16 x y radius	--

R_P2D_Poly(uint8_t nb_Point, point_st *points)	
P2D_Poly(&points, nb_points);	
x17 Nb points point#1.x point#1.y ... point#n.x point#n.y	--

R_P2D_FillPoly(uint8_t nb_Point, point_st *points)			
P2D_FillPoly(&points, nb_points);			
x18	Nb points	point#1.x point#1.y ... point#n.x point#n.y	--
R_P2D_SpriteGetWidth(gui_img_t sprite_id)			
w = SpriteGetWidth(sprite_id); TxMsgPut(w);			
x19	Sprite #ID		width
R_P2D_SpriteGetHeight(gui_img_t sprite_id)			
h = SpriteGetHeight(sprite_id); TxMsgPut(h);			
x1A	Sprite #ID		height
R_SetLut(uint8_t clut_id)			
SetLut(clut_id);			
x1B	CLUT #ID		--
R_Sprite(coord_t x, coord_t y, gui_img_t sprite_id)			
Sprite(x, y, sprite_id);			
x1C	x y	Sprite #ID	--
reserved			
reserved			
x1D			--
reserved			
reserved			
x1E			--
reserved			
reserved			
x1F			--
R_GUI_GetLastAddedObject(uint16_t offset)			
ssm[offset] = GUI_GetLastAddedObject();			
x20	offset		--
R_GUI_ReadSignal(void)			
TxMsgPut(GUI_ReadLastSignal());			
x21			signal
R_GUI_SetAlign(gui_align_t align)			
GUI_SetAlign(align);			
x22	align		--
R_GUI_SetColor(uint16_t color_id, color_t color)			
SetColor(color_id, color)			
x23	Color #ID	color	--

R_GUI_ClearAll(void)	
GUI_ClearAll();	
x24	--
R_GUI_SetGroup(group_t group)	
GUI_SetGroup(group);	
x25 group	--
R_GUI_SetSignal(event_e event, signal_t signal)	
GUI_SetSignal(event, signal);	
x26 event signal	--
R_GUI_GroupDisable(group_t group, bool bDisable)	
GUI_GroupDisable(group, bDisable);	
x27 group bDisable	--
R_GUI_GroupNotify(group_t group, bool bNotify)	
GUI_GroupNotify(group, bNotify);	
x28 group bNotify	--
R_GUI_GroupRefresh(group_t group, bool bRefresh)	
GUI_GroupRefresh(group, bRefresh);	
x29 group bRefresh	--
R_GUI_W_ButtonSetPlacementOrder(btn_placeOrder_e align)	
GUI_W_ButtonSetPlacementOrder(align);	
x2A align	--
R_GUI_W_ButtonAdd(rect_st *rec, void *text, gui_img_t sprite_id)	
GUI_W_ButtonAdd(&rec, text, sprite_id);	
x2B rec.x rec.y rec.w rec.h text Sprite #ID	--
R_GUI_W_CheckBoxAdd(rect_st *rec, void *text, uint16_t offset)	
GUI_W_CheckBoxAdd(&rec, text, &ssm[offset]);	
x2C rec.x rec.y rec.w rec.h text offset	--
R_GUI_W_FrameAdd(rect_st *rec, void *text)	
GUI_W_FrameAdd(&rec, text);	
x2D rec.x rec.y rec.w rec.h text	--
R_GUI_W_GraphSetGridSpacing(length_t hSpacing, length_t vSpacing)	
GUI_W_GraphSetGridSpacing(hSpacing, vSpacing);	
x2E h spacing v spacing	--
R_GUI_W_GraphAdd(rect_st *rec, uint8_t grid, uint16_t refresh_tm)	
GUI_W_GraphAdd(&rec, grid, refresh_tm);	
x2F rec.x rec.y rec.w rec.h grid Refresh tm	--

R_GUI_W_GraphAddCurveToGraph(uint16_t offset, color_t color)	
GUI_W_GraphAddCurveToGraph(NULL, &ssm[offset], color);	
x30 offset color	--

R_GUI_W_GraphAddSampleToCurve(uint16_t offset_obj, uint8_t curve_id, uint8_t sample)	
GUI_W_GraphAddSampleToCurve(ssm[offset_obj], curve_id, sample);	
x31 offset OBJ curve #id sample	--

R_GUI_W_LedAdd(rect_st *rec, void *text, uint16_t offset)	
GUI_W_LedAdd(&rec, text, &ssm[offset]);	
x32 rec.x rec.y rec.w rec.h text offset	--

R_GUI_W_RadioAdd(rect_st *rec, void *text, uint16_t offset, uint8_t id)	
GUI_W_RadioAdd(&rec, text, &ssm[offset], id);	
x33 rec.x rec.y rec.w rec.h text offset #id	--

R_GUI_W_RotaryButtonAdd(rect_st *rec, uint16_t offset, gr_enum_e granularity)	
GUI_W_RotaryButtonAdd(&rec, &ssm[offset], granularity);	
x34 rec.x rec.y rec.w rec.h offset granularity	--

R_GUI_W_RotaryValueAdd(rect_st *rec, uint16_t off_I32, uint16_t off_I8, void *text)	
GUI_W_RotaryValueAdd(&rec, &ssm[off_I32], &ssm[off_I8], text);	
x35 rec.x rec.y rec.w rec.h offset i32 offset i8 text	--

R_GUI_W_RotaryValueLock(uint16_t offset_obj, bool bLock)	
GUI_W_RotaryValueLock(ssm[offset_obj], bLock);	
x36 offset OBJ bLock	--

R_GUI_W_RotaryValueSetMinMax(int32_t min, int32_t max)	
GUI_W_RotaryValueSetMinMax(NULL, min, max);	
x37 min max	--

R_GUI_W_RotaryValueSetDotPos(uint8_t pos_dot)	
GUI_W_RotaryValueSetDotPos(NULL, pos_dot);	
x38 Pos. dot	--

reserved	
reserved	
x39	--

R_GUI_W_SliderAdd(rect_st *rec, uint16_t offset)	
GUI_W_SliderAdd(&rec, &ssm[offset]);	
x3A rec.x rec.y rec.w rec.h offset	--

R_GUI_W_TextAdd(rect_st *rec, void *text)	
GUI_W_TextAdd(&rec, text);	
x3B	rec.x rec.y rec.w rec.h text --

R_GUI_W_UsrEntryAdd (rect_st *rec, uint16_t offset, uint16_t max_len)	
GUI_W_UsrEntryAdd(&rec, &ssm[offset], max_len, false);	
x3C	rec.x rec.y rec.w rec.h offset Max. len --

R_GUI_W_ValueBoxAdd(rect_st *rec, uint16_t offset, valueBoxType_e type, void *text)	
GUI_W_ValueBoxAdd(&rec, &ssm[offset], type, text);	
x3D	rec.x rec.y rec.w rec.h offset type text --

R_GUI_M_FileBrowser(file_browser_mode_e mode, uint16_t offset, uint16_t max_len)	
GUI_M_FileBrowser(mode, &ssm[offset], max_len);	
x3E	mode offset Max. len --

R_GUI_M_KeyboardCreate(coord_t x, coord_t y, kbd_type_e type, uint16_t offsetEntry, uint16_t len, signal_t sigOk, signal_t sigEsc)	
GUI_M_KeyboardCreate(x, y, type, &ssm[offset], max_len, sig_OK, sig_ESC);	
x3F	x y type offset Max. len Sig. OK Sig. ESC --

R_GUI_M_ListAdd(rect_st *rec, bool bHeader, bool bvscroll, bool bhscroll, event_e event, signal_t sig)	
GUI_M_ListAdd(&rec, bHeader, bvscroll, bhscroll, event, signal);	
x40	rec.x rec.y rec.w rec.h bHeader bVscroll bHscroll event signal --

R_GUI_M_ListAddCategoryToList(length_t width, void *text)	
GUI_M_ListAddCategoryToList(width, text);	
x41	width text --

R_GUI_M_ListAddItemToList(gui_img_t sprite_id)	
GUI_M_ListAddItemToList(sprite_id, NULL);	
x42	Sprite #ID --

R_GUI_M_ListAddFieldToLastItem(void *text)	
GUI_M_ListAddFieldToLastItem(text);	
x43	text --

R_GUI_M_ListDeleteContent(void)	
GUI_M_ListDeleteContent();	
x44	--

R_GUI_M_ListGetSelectedItemUid(void)	
GUI_M_ListGetSelectedItemUid(&uid); TxMsgPut(uid);	
x45	Item UID

R_GUI_M_ListGetSelectedItemField(uint8_t catId, uint16_t offset, uint16_t max_len)				
GUI_M_ListGetSelectedItemField(cat_id, &ssm[offset], max_len);				
x46	Cat. #id	offset	Max. len	--

R_GUI_M_Popup(uint8_t btnLeft, uint8_t btnMiddle, uint8_t btnRight, gui_img_t sprite_id, void *text)				
GUI_M_Popup(btnLeft, btnMiddle, btnRight, sprite_id, text);				
x47	Btn left	Btn middle	Btn right	Sprite #ID
				text
				--

R_MEM_WriteU8(uint16_t offset, uint8_t u8)				
SSM_WriteU8(offset, u8);				
x48	offset	u8		--

R_MEM_WriteU16(uint16_t offset, uint16_t u16)				
SSM_WriteU16(offset, u16);				
x49	offset	u16		--

R_MEM_WriteI16(uint16_t offset, int16_t i16)				
SSM_WriteI16(offset, i16);				
x4A	offset	i16		--

R_MEM_WriteU32(uint16_t offset, uint32_t u32)				
SSM_WriteU32(offset, u32);				
x4B	offset	u32		--

R_MEM_WriteI32(uint16_t offset, int32_t i32)				
SSM_WriteI32(offset, i32);				
x4C	offset	i32		--

R_MEM_WriteU8Arr(uint16_t offset, uint16_t nb_bytes, uint8_t *bytes)				
SSM_WriteU8Arr(offset, nb_bytes, &bytes[0]);				
x4D	offset	Nb bytes	u8.1	...
			u8.n	--

R_MEM_WriteStr(uint16_t offset, uint16_t max_len, void *str)				
SSM_WriteStr(offset, max_len, str);				
x4E	offset	Max. len	str	--

R_MEM_Memset(uint16_t offset, uint16_t nb_bytes, uint8_t fill)				
SSM_Memset(offset, nb_bytes, fill);				
x4F	offset	Nb bytes	fill	--

R_MEM_ReadU8(uint16_t offset)				
uint8_t data; SSM_ReadU8(offset, &data); TxMsgPut(data);				
x50	offset			u8

R_MEM_ReadU16(uint16_t offset)				
uint16_t data; SSM_ReadU16(offset, &data); TxMsgPut(data);				
x51	offset			u16

R_MEM_ReadI16(uint16_t offset)					
int16_t data; SSM_ReadI16(offset, &data); TxMsgPut(data);					
x52	offset				i16

R_MEM_ReadU32(uint16_t offset)					
uint32_t data; SSM_ReadU32(offset, &data); TxMsgPut(data);					
x53	offset				u32

R_MEM_ReadI32(uint16_t offset)					
int32_t data; SSM_ReadI32(offset, &data); TxMsgPut(data);					
x54	offset				i32

R_MEM_ReadU8Arr(uint16_t offset, uint16_t nb_bytes)					
while(nb_bytes--) {SSM_ReadU8(offset, &data); TxMsgPut(data);}					
x55	offset	Nb bytes	u8.1	...	u8.n

R_MEM_ReadStr(uint16_t offset, uint16_t max_len)					
while(max_len--) {SSM_ReadU8(offset, &data); TxMsgPut(data); if(data == 0) break;}					
x56	offset	Max. len			str

10.Portability

DRAFT