/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  Arduino TFT graphics library targeted at ESP8266

  and ESP32 based boards.

  This is a stand-alone library that contains the

  hardware driver, the graphics functions and the

  proportional fonts.

  The built-in fonts 4, 6, 7 and 8 are Run Length

  Encoded (RLE) to reduce the FLASH footprint.

  Last review/edit by Bodmer: 04/02/22

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Stop fonts etc being loaded multiple times

#ifndef \_TFT\_eSPIH\_

#define \_TFT\_eSPIH\_

#define TFT\_ESPI\_VERSION "2.4.79"

// Bit level feature flags

// Bit 0 set: viewport capability

#define TFT\_ESPI\_FEATURES 1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 1: Load required header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Standard support

#include <Arduino.h>

#include <Print.h>

#include <SPI.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 2: Load library and processor specific header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Include header file that defines the fonts loaded, the TFT drivers

// available and the pins to be used, etc, etc

#ifdef CONFIG\_TFT\_eSPI\_ESPIDF

  #include "TFT\_config.h"

#endif

// New ESP8266 board package uses ARDUINO\_ARCH\_ESP8266

// old package defined ESP8266

#if defined (ESP8266)

  #ifndef ARDUINO\_ARCH\_ESP8266

    #define ARDUINO\_ARCH\_ESP8266

  #endif

#endif

// The following lines allow the user setup to be included in the sketch folder, see

// "Sketch\_with\_tft\_setup" generic example.

#if !defined \_\_has\_include

  #if !defined(DISABLE\_ALL\_LIBRARY\_WARNINGS)

    #warning Compiler does not support \_\_has\_include, so sketches cannot define the setup

  #endif

#else

  #if \_\_has\_include(<tft\_setup.h>)

    // Include the sketch setup file

    #include <tft\_setup.h>

    #ifndef USER\_SETUP\_LOADED

      // Prevent loading further setups

      #define USER\_SETUP\_LOADED

    #endif

  #endif

#endif

#include <User\_Setup\_Select.h>

// Handle FLASH based storage e.g. PROGMEM

#if defined(ARDUINO\_ARCH\_RP2040)

  #undef pgm\_read\_byte

  #define pgm\_read\_byte(addr)   (\*(const unsigned char \*)(addr))

  #undef pgm\_read\_word

  #define pgm\_read\_word(addr) ({ \

    typeof(addr) \_addr = (addr); \

    \*(const unsigned short \*)(\_addr); \

  })

  #undef pgm\_read\_dword

  #define pgm\_read\_dword(addr) ({ \

    typeof(addr) \_addr = (addr); \

    \*(const unsigned long \*)(\_addr); \

  })

#elif defined(\_\_AVR\_\_)

  #include <avr/pgmspace.h>

#elif defined(ARDUINO\_ARCH\_ESP8266) || defined(ESP32)

  #include <pgmspace.h>

#else

  #define PROGMEM

#endif

// Include the processor specific drivers

#if defined(CONFIG\_IDF\_TARGET\_ESP32S3)

  #include "Processors/TFT\_eSPI\_ESP32\_S3.h"

#elif defined(CONFIG\_IDF\_TARGET\_ESP32C3)

  #include "Processors/TFT\_eSPI\_ESP32\_C3.h"

#elif defined (ESP32)

  #include "Processors/TFT\_eSPI\_ESP32.h"

#elif defined (ARDUINO\_ARCH\_ESP8266)

  #include "Processors/TFT\_eSPI\_ESP8266.h"

#elif defined (STM32)

  #include "Processors/TFT\_eSPI\_STM32.h"

#elif defined(ARDUINO\_ARCH\_RP2040)

  #include "Processors/TFT\_eSPI\_RP2040.h"

#else

  #include "Processors/TFT\_eSPI\_Generic.h"

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 3: Interface setup

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef TAB\_COLOUR

  #define TAB\_COLOUR 0

#endif

// If the SPI frequency is not defined, set a default

#ifndef SPI\_FREQUENCY

  #define SPI\_FREQUENCY  20000000

#endif

// If the SPI read frequency is not defined, set a default

#ifndef SPI\_READ\_FREQUENCY

  #define SPI\_READ\_FREQUENCY 10000000

#endif

// Some ST7789 boards do not work with Mode 0

#ifndef TFT\_SPI\_MODE

  #if defined(ST7789\_DRIVER) || defined(ST7789\_2\_DRIVER)

    #define TFT\_SPI\_MODE SPI\_MODE3

  #else

    #define TFT\_SPI\_MODE SPI\_MODE0

  #endif

#endif

// If the XPT2046 SPI frequency is not defined, set a default

#ifndef SPI\_TOUCH\_FREQUENCY

  #define SPI\_TOUCH\_FREQUENCY  2500000

#endif

#ifndef SPI\_BUSY\_CHECK

  #define SPI\_BUSY\_CHECK

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 4: Setup fonts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Use GLCD font in error case where user requests a smooth font file

// that does not exist (this is a temporary fix to stop ESP32 reboot)

#ifdef SMOOTH\_FONT

  #ifndef LOAD\_GLCD

    #define LOAD\_GLCD

  #endif

#endif

// Only load the fonts defined in User\_Setup.h (to save space)

// Set flag so RLE rendering code is optionally compiled

#ifdef LOAD\_GLCD

  #include <Fonts/glcdfont.c>

#endif

#ifdef LOAD\_FONT2

  #include <Fonts/Font16.h>

#endif

#ifdef LOAD\_FONT4

  #include <Fonts/Font32rle.h>

  #define LOAD\_RLE

#endif

#ifdef LOAD\_FONT6

  #include <Fonts/Font64rle.h>

  #ifndef LOAD\_RLE

    #define LOAD\_RLE

  #endif

#endif

#ifdef LOAD\_FONT7

  #include <Fonts/Font7srle.h>

  #ifndef LOAD\_RLE

    #define LOAD\_RLE

  #endif

#endif

#ifdef LOAD\_FONT8

  #include <Fonts/Font72rle.h>

  #ifndef LOAD\_RLE

    #define LOAD\_RLE

  #endif

#elif defined LOAD\_FONT8N // Optional narrower version

  #define LOAD\_FONT8

  #include <Fonts/Font72x53rle.h>

  #ifndef LOAD\_RLE

    #define LOAD\_RLE

  #endif

#endif

#ifdef LOAD\_GFXFF

  // We can include all the free fonts and they will only be built into

  // the sketch if they are used

  #include <Fonts/GFXFF/gfxfont.h>

  // Call up any user custom fonts

  #include <User\_Setups/User\_Custom\_Fonts.h>

#endif // #ifdef LOAD\_GFXFF

// Create a null default font in case some fonts not used (to prevent crash)

const  uint8\_t widtbl\_null[1] = {0};

PROGMEM const uint8\_t chr\_null[1] = {0};

PROGMEM const uint8\_t\* const chrtbl\_null[1] = {chr\_null};

// This is a structure to conveniently hold information on the default fonts

// Stores pointer to font character image address table, width table and height

typedef struct {

    const uint8\_t \*chartbl;

    const uint8\_t \*widthtbl;

    uint8\_t height;

    uint8\_t baseline;

    } fontinfo;

// Now fill the structure

const PROGMEM fontinfo fontdata [] = {

  #ifdef LOAD\_GLCD

   { (const uint8\_t \*)font, widtbl\_null, 0, 0 },

  #else

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 },

  #endif

   // GLCD font (Font 1) does not have all parameters

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 8, 7 },

  #ifdef LOAD\_FONT2

   { (const uint8\_t \*)chrtbl\_f16, widtbl\_f16, chr\_hgt\_f16, baseline\_f16},

  #else

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 },

  #endif

   // Font 3 current unused

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 },

  #ifdef LOAD\_FONT4

   { (const uint8\_t \*)chrtbl\_f32, widtbl\_f32, chr\_hgt\_f32, baseline\_f32},

  #else

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 },

  #endif

   // Font 5 current unused

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 },

  #ifdef LOAD\_FONT6

   { (const uint8\_t \*)chrtbl\_f64, widtbl\_f64, chr\_hgt\_f64, baseline\_f64},

  #else

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 },

  #endif

  #ifdef LOAD\_FONT7

   { (const uint8\_t \*)chrtbl\_f7s, widtbl\_f7s, chr\_hgt\_f7s, baseline\_f7s},

  #else

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 },

  #endif

  #ifdef LOAD\_FONT8

   { (const uint8\_t \*)chrtbl\_f72, widtbl\_f72, chr\_hgt\_f72, baseline\_f72}

  #else

   { (const uint8\_t \*)chrtbl\_null, widtbl\_null, 0, 0 }

  #endif

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 5: Font datum enumeration

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//These enumerate the text plotting alignment (reference datum point)

#define TL\_DATUM 0 // Top left (default)

#define TC\_DATUM 1 // Top centre

#define TR\_DATUM 2 // Top right

#define ML\_DATUM 3 // Middle left

#define CL\_DATUM 3 // Centre left, same as above

#define MC\_DATUM 4 // Middle centre

#define CC\_DATUM 4 // Centre centre, same as above

#define MR\_DATUM 5 // Middle right

#define CR\_DATUM 5 // Centre right, same as above

#define BL\_DATUM 6 // Bottom left

#define BC\_DATUM 7 // Bottom centre

#define BR\_DATUM 8 // Bottom right

#define L\_BASELINE  9 // Left character baseline (Line the 'A' character would sit on)

#define C\_BASELINE 10 // Centre character baseline

#define R\_BASELINE 11 // Right character baseline

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 6: Colour enumeration

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Default color definitions

#define TFT\_BLACK       0x0000      /\*   0,   0,   0 \*/

#define TFT\_NAVY        0x000F      /\*   0,   0, 128 \*/

#define TFT\_DARKGREEN   0x03E0      /\*   0, 128,   0 \*/

#define TFT\_DARKCYAN    0x03EF      /\*   0, 128, 128 \*/

#define TFT\_MAROON      0x7800      /\* 128,   0,   0 \*/

#define TFT\_PURPLE      0x780F      /\* 128,   0, 128 \*/

#define TFT\_OLIVE       0x7BE0      /\* 128, 128,   0 \*/

#define TFT\_LIGHTGREY   0xD69A      /\* 211, 211, 211 \*/

#define TFT\_DARKGREY    0x7BEF      /\* 128, 128, 128 \*/

#define TFT\_BLUE        0x001F      /\*   0,   0, 255 \*/

#define TFT\_GREEN       0x07E0      /\*   0, 255,   0 \*/

#define TFT\_CYAN        0x07FF      /\*   0, 255, 255 \*/

#define TFT\_RED         0xF800      /\* 255,   0,   0 \*/

#define TFT\_MAGENTA     0xF81F      /\* 255,   0, 255 \*/

#define TFT\_YELLOW      0xFFE0      /\* 255, 255,   0 \*/

#define TFT\_WHITE       0xFFFF      /\* 255, 255, 255 \*/

#define TFT\_ORANGE      0xFDA0      /\* 255, 180,   0 \*/

#define TFT\_GREENYELLOW 0xB7E0      /\* 180, 255,   0 \*/

#define TFT\_PINK        0xFE19      /\* 255, 192, 203 \*/ //Lighter pink, was 0xFC9F

#define TFT\_BROWN       0x9A60      /\* 150,  75,   0 \*/

#define TFT\_GOLD        0xFEA0      /\* 255, 215,   0 \*/

#define TFT\_SILVER      0xC618      /\* 192, 192, 192 \*/

#define TFT\_SKYBLUE     0x867D      /\* 135, 206, 235 \*/

#define TFT\_VIOLET      0x915C      /\* 180,  46, 226 \*/

// Next is a special 16 bit colour value that encodes to 8 bits

// and will then decode back to the same 16 bit value.

// Convenient for 8 bit and 16 bit transparent sprites.

#define TFT\_TRANSPARENT 0x0120 // This is actually a dark green

// Default palette for 4 bit colour sprites

static const uint16\_t default\_4bit\_palette[] PROGMEM = {

  TFT\_BLACK,    //  0  ^

  TFT\_BROWN,    //  1  |

  TFT\_RED,      //  2  |

  TFT\_ORANGE,   //  3  |

  TFT\_YELLOW,   //  4  Colours 0-9 follow the resistor colour code!

  TFT\_GREEN,    //  5  |

  TFT\_BLUE,     //  6  |

  TFT\_PURPLE,   //  7  |

  TFT\_DARKGREY, //  8  |

  TFT\_WHITE,    //  9  v

  TFT\_CYAN,     // 10  Blue+green mix

  TFT\_MAGENTA,  // 11  Blue+red mix

  TFT\_MAROON,   // 12  Darker red colour

  TFT\_DARKGREEN,// 13  Darker green colour

  TFT\_NAVY,     // 14  Darker blue colour

  TFT\_PINK      // 15

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 7: Diagnostic support

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// #define TFT\_eSPI\_DEBUG     // Switch on debug support serial messages  (not used yet)

// #define TFT\_eSPI\_FNx\_DEBUG // Switch on debug support for function "x" (not used yet)

// This structure allows sketches to retrieve the user setup parameters at runtime

// by calling getSetup(), zero impact on code size unless used, mainly for diagnostics

typedef struct

{

String  version = TFT\_ESPI\_VERSION;

String  setup\_info;  // Setup reference name available to use in a user setup

uint32\_t setup\_id;   // ID available to use in a user setup

int32\_t esp;         // Processor code

uint8\_t trans;       // SPI transaction support

uint8\_t serial;      // Serial (SPI) or parallel

uint8\_t  port;       // SPI port

uint8\_t overlap;     // ESP8266 overlap mode

uint8\_t interface;   // Interface type

uint16\_t tft\_driver; // Hexadecimal code

uint16\_t tft\_width;  // Rotation 0 width and height

uint16\_t tft\_height;

uint8\_t r0\_x\_offset; // Display offsets, not all used yet

uint8\_t r0\_y\_offset;

uint8\_t r1\_x\_offset;

uint8\_t r1\_y\_offset;

uint8\_t r2\_x\_offset;

uint8\_t r2\_y\_offset;

uint8\_t r3\_x\_offset;

uint8\_t r3\_y\_offset;

int8\_t pin\_tft\_mosi; // SPI pins

int8\_t pin\_tft\_miso;

int8\_t pin\_tft\_clk;

int8\_t pin\_tft\_cs;

int8\_t pin\_tft\_dc;   // Control pins

int8\_t pin\_tft\_rd;

int8\_t pin\_tft\_wr;

int8\_t pin\_tft\_rst;

int8\_t pin\_tft\_d0;   // Parallel port pins

int8\_t pin\_tft\_d1;

int8\_t pin\_tft\_d2;

int8\_t pin\_tft\_d3;

int8\_t pin\_tft\_d4;

int8\_t pin\_tft\_d5;

int8\_t pin\_tft\_d6;

int8\_t pin\_tft\_d7;

int8\_t pin\_tft\_led;

int8\_t pin\_tft\_led\_on;

int8\_t pin\_tch\_cs;   // Touch chip select pin

int16\_t tft\_spi\_freq;// TFT write SPI frequency

int16\_t tft\_rd\_freq; // TFT read  SPI frequency

int16\_t tch\_spi\_freq;// Touch controller read/write SPI frequency

} setup\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 8: Class member and support functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Swap any type

template <typename T> static inline void

swap\_coord(T& a, T& b) { T t = a; a = b; b = t; }

// Callback prototype for smooth font pixel colour read

typedef uint16\_t (\*getColorCallback)(uint16\_t x, uint16\_t y);

// Class functions and variables

class TFT\_eSPI : public Print { friend class TFT\_eSprite; // Sprite class has access to protected members

 //--------------------------------------- public ------------------------------------//

 public:

  TFT\_eSPI(int16\_t \_W = TFT\_WIDTH, int16\_t \_H = TFT\_HEIGHT);

  // init() and begin() are equivalent, begin() included for backwards compatibility

  // Sketch defined tab colour option is for ST7735 displays only

  void     init(uint8\_t tc = TAB\_COLOUR), begin(uint8\_t tc = TAB\_COLOUR);

  // These are virtual so the TFT\_eSprite class can override them with sprite specific functions

  virtual void     drawPixel(int32\_t x, int32\_t y, uint32\_t color),

                   drawChar(int32\_t x, int32\_t y, uint16\_t c, uint32\_t color, uint32\_t bg, uint8\_t size),

                   drawLine(int32\_t xs, int32\_t ys, int32\_t xe, int32\_t ye, uint32\_t color),

                   drawFastVLine(int32\_t x, int32\_t y, int32\_t h, uint32\_t color),

                   drawFastHLine(int32\_t x, int32\_t y, int32\_t w, uint32\_t color),

                   fillRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint32\_t color);

  virtual int16\_t  drawChar(uint16\_t uniCode, int32\_t x, int32\_t y, uint8\_t font),

                   drawChar(uint16\_t uniCode, int32\_t x, int32\_t y),

                   height(void),

                   width(void);

                   // Read the colour of a pixel at x,y and return value in 565 format

  virtual uint16\_t readPixel(int32\_t x, int32\_t y);

  virtual void     setWindow(int32\_t xs, int32\_t ys, int32\_t xe, int32\_t ye);   // Note: start + end coordinates

                   // Push (aka write pixel) colours to the set window

  virtual void     pushColor(uint16\_t color);

                   // These are non-inlined to enable override

  virtual void     begin\_nin\_write();

  virtual void     end\_nin\_write();

  void     setRotation(uint8\_t r); // Set the display image orientation to 0, 1, 2 or 3

  uint8\_t  getRotation(void);      // Read the current rotation

  void     invertDisplay(bool i);  // Tell TFT to invert all displayed colours

  // The TFT\_eSprite class inherits the following functions (not all are useful to Sprite class

  void     setAddrWindow(int32\_t xs, int32\_t ys, int32\_t w, int32\_t h); // Note: start coordinates + width and height

  // Viewport commands, see "Viewport\_Demo" sketch

  void     setViewport(int32\_t x, int32\_t y, int32\_t w, int32\_t h, bool vpDatum = true);

  bool     checkViewport(int32\_t x, int32\_t y, int32\_t w, int32\_t h);

  int32\_t  getViewportX(void);

  int32\_t  getViewportY(void);

  int32\_t  getViewportWidth(void);

  int32\_t  getViewportHeight(void);

  bool     getViewportDatum(void);

  void     frameViewport(uint16\_t color, int32\_t w);

  void     resetViewport(void);

           // Clip input window to viewport bounds, return false if whole area is out of bounds

  bool     clipAddrWindow(int32\_t\* x, int32\_t\* y, int32\_t\* w, int32\_t\* h);

           // Clip input window area to viewport bounds, return false if whole area is out of bounds

  bool     clipWindow(int32\_t\* xs, int32\_t\* ys, int32\_t\* xe, int32\_t\* ye);

           // Push (aka write pixel) colours to the TFT (use setAddrWindow() first)

  void     pushColor(uint16\_t color, uint32\_t len),  // Deprecated, use pushBlock()

           pushColors(uint16\_t  \*data, uint32\_t len, bool swap = true), // With byte swap option

           pushColors(uint8\_t  \*data, uint32\_t len); // Deprecated, use pushPixels()

           // Write a solid block of a single colour

  void     pushBlock(uint16\_t color, uint32\_t len);

           // Write a set of pixels stored in memory, use setSwapBytes(true/false) function to correct endianess

  void     pushPixels(const void \* data\_in, uint32\_t len);

           // Support for half duplex (bi-directional SDA) SPI bus where MOSI must be switched to input

           #ifdef TFT\_SDA\_READ

             #if defined (TFT\_eSPI\_ENABLE\_8\_BIT\_READ)

  uint8\_t  tft\_Read\_8(void);     // Read 8 bit value from TFT command register

             #endif

  void     begin\_SDA\_Read(void); // Begin a read on a half duplex (bi-directional SDA) SPI bus - sets MOSI to input

  void     end\_SDA\_Read(void);   // Restore MOSI to output

           #endif

  // Graphics drawing

  void     fillScreen(uint32\_t color),

           drawRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint32\_t color),

           drawRoundRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, int32\_t radius, uint32\_t color),

           fillRoundRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, int32\_t radius, uint32\_t color);

  void     fillRectVGradient(int16\_t x, int16\_t y, int16\_t w, int16\_t h, uint32\_t color1, uint32\_t color2);

  void     fillRectHGradient(int16\_t x, int16\_t y, int16\_t w, int16\_t h, uint32\_t color1, uint32\_t color2);

           // Draw a pixel blended with the pixel colour on the TFT or sprite, return blended colour

           // If bg\_color is not included the background pixel colour will be read from TFT or sprite

  uint16\_t drawPixel(int32\_t x, int32\_t y, uint32\_t color, uint8\_t alpha, uint32\_t bg\_color = 0x00FFFFFF);

           // Draw a small anti-aliased filled circle at ax,ay with radius r (uses drawWideLine)

           // If bg\_color is not included the background pixel colour will be read from TFT or sprite

  void     drawSpot(float ax, float ay, float r, uint32\_t fg\_color, uint32\_t bg\_color = 0x00FFFFFF);

           // Draw an anti-aliased filled circle at x, y with radius r

           // If bg\_color is not included the background pixel colour will be read from TFT or sprite

  void     fillSmoothCircle(int32\_t x, int32\_t y, int32\_t r, uint32\_t color, uint32\_t bg\_color = 0x00FFFFFF);

  void     fillSmoothRoundRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, int32\_t radius, uint32\_t color, uint32\_t bg\_color = 0x00FFFFFF);

           // Draw an anti-aliased wide line from ax,ay to bx,by width wd with radiused ends (radius is wd/2)

           // If bg\_color is not included the background pixel colour will be read from TFT or sprite

  void     drawWideLine(float ax, float ay, float bx, float by, float wd, uint32\_t fg\_color, uint32\_t bg\_color = 0x00FFFFFF);

           // Draw an anti-aliased wide line from ax,ay to bx,by with different width at each end aw, bw and with radiused ends

           // If bg\_color is not included the background pixel colour will be read from TFT or sprite

  void     drawWedgeLine(float ax, float ay, float bx, float by, float aw, float bw, uint32\_t fg\_color, uint32\_t bg\_color = 0x00FFFFFF);

  void     drawCircle(int32\_t x, int32\_t y, int32\_t r, uint32\_t color),

           drawCircleHelper(int32\_t x, int32\_t y, int32\_t r, uint8\_t cornername, uint32\_t color),

           fillCircle(int32\_t x, int32\_t y, int32\_t r, uint32\_t color),

           fillCircleHelper(int32\_t x, int32\_t y, int32\_t r, uint8\_t cornername, int32\_t delta, uint32\_t color),

           drawEllipse(int16\_t x, int16\_t y, int32\_t rx, int32\_t ry, uint16\_t color),

           fillEllipse(int16\_t x, int16\_t y, int32\_t rx, int32\_t ry, uint16\_t color),

           //                 Corner 1               Corner 2               Corner 3

           drawTriangle(int32\_t x1,int32\_t y1, int32\_t x2,int32\_t y2, int32\_t x3,int32\_t y3, uint32\_t color),

           fillTriangle(int32\_t x1,int32\_t y1, int32\_t x2,int32\_t y2, int32\_t x3,int32\_t y3, uint32\_t color);

  // Image rendering

           // Swap the byte order for pushImage() and pushPixels() - corrects endianness

  void     setSwapBytes(bool swap);

  bool     getSwapBytes(void);

           // Draw bitmap

  void     drawBitmap( int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t fgcolor),

           drawBitmap( int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t fgcolor, uint16\_t bgcolor),

           drawXBitmap(int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t fgcolor),

           drawXBitmap(int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t fgcolor, uint16\_t bgcolor),

           setBitmapColor(uint16\_t fgcolor, uint16\_t bgcolor); // Define the 2 colours for 1bpp sprites

           // Set TFT pivot point (use when rendering rotated sprites)

  void     setPivot(int16\_t x, int16\_t y);

  int16\_t  getPivotX(void), // Get pivot x

           getPivotY(void); // Get pivot y

           // The next functions can be used as a pair to copy screen blocks (or horizontal/vertical lines) to another location

           // Read a block of pixels to a data buffer, buffer is 16 bit and the size must be at least w \* h

  void     readRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data);

           // Write a block of pixels to the screen which have been read by readRect()

  void     pushRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data);

           // These are used to render images or sprites stored in RAM arrays (used by Sprite class for 16bpp Sprites)

  void     pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data);

  void     pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data, uint16\_t transparent);

           // These are used to render images stored in FLASH (PROGMEM)

  void     pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, const uint16\_t \*data, uint16\_t transparent);

  void     pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, const uint16\_t \*data);

           // These are used by Sprite class pushSprite() member function for 1, 4 and 8 bits per pixel (bpp) colours

           // They are not intended to be used with user sketches (but could be)

           // Set bpp8 true for 8bpp sprites, false otherwise. The cmap pointer must be specified for 4bpp

  void     pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint8\_t  \*data, bool bpp8 = true, uint16\_t \*cmap = nullptr);

  void     pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint8\_t  \*data, uint8\_t  transparent, bool bpp8 = true, uint16\_t \*cmap = nullptr);

           // FLASH version

  void     pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, const uint8\_t \*data, bool bpp8,  uint16\_t \*cmap = nullptr);

           // This next function has been used successfully to dump the TFT screen to a PC for documentation purposes

           // It reads a screen area and returns the 3 RGB 8 bit colour values of each pixel in the buffer

           // Set w and h to 1 to read 1 pixel's colour. The data buffer must be at least w \* h \* 3 bytes

  void     readRectRGB(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint8\_t \*data);

  // Text rendering - value returned is the pixel width of the rendered text

  int16\_t  drawNumber(long intNumber, int32\_t x, int32\_t y, uint8\_t font), // Draw integer using specified font number

           drawNumber(long intNumber, int32\_t x, int32\_t y),               // Draw integer using current font

           // Decimal is the number of decimal places to render

           // Use with setTextDatum() to position values on TFT, and setTextPadding() to blank old displayed values

           drawFloat(float floatNumber, uint8\_t decimal, int32\_t x, int32\_t y, uint8\_t font), // Draw float using specified font number

           drawFloat(float floatNumber, uint8\_t decimal, int32\_t x, int32\_t y),               // Draw float using current font

           // Handle char arrays

           // Use with setTextDatum() to position string on TFT, and setTextPadding() to blank old displayed strings

           drawString(const char \*string, int32\_t x, int32\_t y, uint8\_t font),  // Draw string using specified font number

           drawString(const char \*string, int32\_t x, int32\_t y),                // Draw string using current font

           drawString(const String& string, int32\_t x, int32\_t y, uint8\_t font),// Draw string using specified font number

           drawString(const String& string, int32\_t x, int32\_t y),              // Draw string using current font

           drawCentreString(const char \*string, int32\_t x, int32\_t y, uint8\_t font),  // Deprecated, use setTextDatum() and drawString()

           drawRightString(const char \*string, int32\_t x, int32\_t y, uint8\_t font),   // Deprecated, use setTextDatum() and drawString()

           drawCentreString(const String& string, int32\_t x, int32\_t y, uint8\_t font),// Deprecated, use setTextDatum() and drawString()

           drawRightString(const String& string, int32\_t x, int32\_t y, uint8\_t font); // Deprecated, use setTextDatum() and drawString()

  // Text rendering and font handling support funtions

  void     setCursor(int16\_t x, int16\_t y),                 // Set cursor for tft.print()

           setCursor(int16\_t x, int16\_t y, uint8\_t font);   // Set cursor and font number for tft.print()

  int16\_t  getCursorX(void),                                // Read current cursor x position (moves with tft.print())

           getCursorY(void);                                // Read current cursor y position

  void     setTextColor(uint16\_t color),                    // Set character (glyph) color only (background not over-written)

           setTextColor(uint16\_t fgcolor, uint16\_t bgcolor, bool bgfill = false),  // Set character (glyph) foreground and background colour, optional background fill for smooth fonts

           setTextSize(uint8\_t size);                       // Set character size multiplier (this increases pixel size)

  void     setTextWrap(bool wrapX, bool wrapY = false);     // Turn on/off wrapping of text in TFT width and/or height

  void     setTextDatum(uint8\_t datum);                     // Set text datum position (default is top left), see Section 6 above

  uint8\_t  getTextDatum(void);

  void     setTextPadding(uint16\_t x\_width);                // Set text padding (background blanking/over-write) width in pixels

  uint16\_t getTextPadding(void);                            // Get text padding

#ifdef LOAD\_GFXFF

  void     setFreeFont(const GFXfont \*f = NULL),            // Select the GFX Free Font

           setTextFont(uint8\_t font);                       // Set the font number to use in future

#else

  void     setFreeFont(uint8\_t font),                       // Not used, historical fix to prevent an error

           setTextFont(uint8\_t font);                       // Set the font number to use in future

#endif

  int16\_t  textWidth(const char \*string, uint8\_t font),     // Returns pixel width of string in specified font

           textWidth(const char \*string),                   // Returns pixel width of string in current font

           textWidth(const String& string, uint8\_t font),   // As above for String types

           textWidth(const String& string),

           fontHeight(int16\_t font),                        // Returns pixel height of string in specified font

           fontHeight(void);                                // Returns pixel width of string in current font

           // Used by library and Smooth font class to extract Unicode point codes from a UTF8 encoded string

  uint16\_t decodeUTF8(uint8\_t \*buf, uint16\_t \*index, uint16\_t remaining),

           decodeUTF8(uint8\_t c);

           // Support function to UTF8 decode and draw characters piped through print stream

  size\_t   write(uint8\_t);

  // size\_t   write(const uint8\_t \*buf, size\_t len);

           // Used by Smooth font class to fetch a pixel colour for the anti-aliasing

  void     setCallback(getColorCallback getCol);

  uint16\_t fontsLoaded(void); // Each bit in returned value represents a font type that is loaded - used for debug/error handling only

  // Low level read/write

  void     spiwrite(uint8\_t);        // legacy support only

#ifndef RM68120\_DRIVER

  void     writecommand(uint8\_t c);  // Send a command, function resets DC/RS high ready for data

#else

  void     writecommand(uint16\_t c); // Send a command, function resets DC/RS high ready for data

  void     writeRegister(uint16\_t c, uint8\_t d); // Write data to 16 bit command register

#endif

  void     writedata(uint8\_t d);     // Send data with DC/RS set high

  void     commandList(const uint8\_t \*addr); // Send a initialisation sequence to TFT stored in FLASH

  uint8\_t  readcommand8( uint8\_t cmd\_function, uint8\_t index = 0); // read 8 bits from TFT

  uint16\_t readcommand16(uint8\_t cmd\_function, uint8\_t index = 0); // read 16 bits from TFT

  uint32\_t readcommand32(uint8\_t cmd\_function, uint8\_t index = 0); // read 32 bits from TFT

  // Colour conversion

           // Convert 8 bit red, green and blue to 16 bits

  uint16\_t color565(uint8\_t red, uint8\_t green, uint8\_t blue);

           // Convert 8 bit colour to 16 bits

  uint16\_t color8to16(uint8\_t color332);

           // Convert 16 bit colour to 8 bits

  uint8\_t  color16to8(uint16\_t color565);

           // Convert 16 bit colour to/from 24 bit, R+G+B concatenated into LS 24 bits

  uint32\_t color16to24(uint16\_t color565);

  uint32\_t color24to16(uint32\_t color888);

           // Alpha blend 2 colours, see generic "alphaBlend\_Test" example

           // alpha =   0 = 100% background colour

           // alpha = 255 = 100% foreground colour

  uint16\_t alphaBlend(uint8\_t alpha, uint16\_t fgc, uint16\_t bgc);

           // 16 bit colour alphaBlend with alpha dither (dither reduces colour banding)

  uint16\_t alphaBlend(uint8\_t alpha, uint16\_t fgc, uint16\_t bgc, uint8\_t dither);

           // 24 bit colour alphaBlend with optional alpha dither

  uint32\_t alphaBlend24(uint8\_t alpha, uint32\_t fgc, uint32\_t bgc, uint8\_t dither = 0);

  // DMA support functions - these are currently just for SPI writes when using the ESP32 or STM32 processors

  // DMA works also on RP2040 and PIO SPI, 8 bit parallel and 16 bit parallel

           // Bear in mind DMA will only be of benefit in particular circumstances and can be tricky

           // to manage by noobs. The functions have however been designed to be noob friendly and

           // avoid a few DMA behaviour "gotchas".

           //

           // At best you will get a 2x TFT rendering performance improvement when using DMA because

           // this library handles the SPI bus so efficiently during normal (non DMA) transfers. The best

           // performance improvement scenario is the DMA transfer time is exactly the same as the time it

           // takes for the processor to prepare the next image buffer and initiate another DMA transfer.

           //

           // DMA transfer to the TFT is done while the processor moves on to handle other tasks. Bear

           // this in mind and watch out for "gotchas" like the image buffer going out of scope as the

           // processor leaves a function or its content being changed while the DMA engine is reading it.

           //

           // The compiler MAY change the implied scope of a buffer which has been set aside by creating

           // an array. For example a buffer defined before a "for-next" loop may get de-allocated when

           // the loop ends. To avoid this use, for example, malloc() and free() to take control of when

           // the buffer space is available and ensure it is not released until DMA is complete.

           //

           // Clearly you should not modify a buffer that is being DMA'ed to the TFT until the DMA is over.

           // Use the dmaBusy() function to check this.  Use tft.startWrite() before invoking DMA so the

           // TFT chip select stays low. If you use tft.endWrite() before DMA is complete then the endWrite

           // function will wait for the DMA to complete, so this may defeat any DMA performance benefit.

           //

  bool     initDMA(bool ctrl\_cs = false);  // Initialise the DMA engine and attach to SPI bus - typically used in setup()

                                           // Parameter "true" enables DMA engine control of TFT chip select (ESP32 only)

                                           // For ESP32 only, TFT reads will not work if parameter is true

  void     deInitDMA(void);   // De-initialise the DMA engine and detach from SPI bus - typically not used

           // Push an image to the TFT using DMA, buffer is optional and grabs (double buffers) a copy of the image

           // Use the buffer if the image data will get over-written or destroyed while DMA is in progress

           // If swapping colour bytes is defined, and the double buffer option is NOT used, then the bytes

           // in the original data image will be swapped by the function before DMA is initiated.

           // The function will wait for the last DMA to complete if it is called while a previous DMA is still

           // in progress, this simplifies the sketch and helps avoid "gotchas".

  void     pushImageDMA(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t\* data, uint16\_t\* buffer = nullptr);

#if defined (ESP32) // ESP32 only at the moment

           // For case where pointer is a const and the image data must not be modified (clipped or byte swapped)

  void     pushImageDMA(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t const\* data);

#endif

           // Push a block of pixels into a window set up using setAddrWindow()

  void     pushPixelsDMA(uint16\_t\* image, uint32\_t len);

           // Check if the DMA is complete - use while(tft.dmaBusy); for a blocking wait

  bool     dmaBusy(void); // returns true if DMA is still in progress

  void     dmaWait(void); // wait until DMA is complete

  bool     DMA\_Enabled = false;   // Flag for DMA enabled state

  uint8\_t  spiBusyCheck = 0;      // Number of ESP32 transfer buffers to check

  // Bare metal functions

  void     startWrite(void);                         // Begin SPI transaction

  void     writeColor(uint16\_t color, uint32\_t len); // Deprecated, use pushBlock()

  void     endWrite(void);                           // End SPI transaction

  // Set/get an arbitrary library configuration attribute or option

  //       Use to switch ON/OFF capabilities such as UTF8 decoding - each attribute has a unique ID

  //       id = 0: reserved - may be used in future to reset all attributes to a default state

  //       id = 1: Turn on (a=true) or off (a=false) GLCD cp437 font character error correction

  //       id = 2: Turn on (a=true) or off (a=false) UTF8 decoding

  //       id = 3: Enable or disable use of ESP32 PSRAM (if available)

           #define CP437\_SWITCH 1

           #define UTF8\_SWITCH  2

           #define PSRAM\_ENABLE 3

  void     setAttribute(uint8\_t id = 0, uint8\_t a = 0); // Set attribute value

  uint8\_t  getAttribute(uint8\_t id = 0);                // Get attribute value

           // Used for diagnostic sketch to see library setup adopted by compiler, see Section 7 above

  void     getSetup(setup\_t& tft\_settings); // Sketch provides the instance to populate

  bool     verifySetupID(uint32\_t id);

  // Global variables

  static   SPIClass& getSPIinstance(void); // Get SPI class handle

  uint32\_t textcolor, textbgcolor;         // Text foreground and background colours

  uint32\_t bitmap\_fg, bitmap\_bg;           // Bitmap foreground (bit=1) and background (bit=0) colours

  uint8\_t  textfont,  // Current selected font number

           textsize,  // Current font size multiplier

           textdatum, // Text reference datum

           rotation;  // Display rotation (0-3)

  uint8\_t  decoderState = 0;   // UTF8 decoder state        - not for user access

  uint16\_t decoderBuffer;      // Unicode code-point buffer - not for user access

 //--------------------------------------- private ------------------------------------//

 private:

           // Legacy begin and end prototypes - deprecated TODO: delete

  void     spi\_begin();

  void     spi\_end();

  void     spi\_begin\_read();

  void     spi\_end\_read();

           // New begin and end prototypes

           // begin/end a TFT write transaction

           // For SPI bus the transmit clock rate is set

  inline void begin\_tft\_write() \_\_attribute\_\_((always\_inline));

  inline void end\_tft\_write()   \_\_attribute\_\_((always\_inline));

           // begin/end a TFT read transaction

           // For SPI bus: begin lowers SPI clock rate, end reinstates transmit clock rate

  inline void begin\_tft\_read()  \_\_attribute\_\_((always\_inline));

  inline void end\_tft\_read()    \_\_attribute\_\_((always\_inline));

           // Initialise the data bus GPIO and hardware interfaces

  void     initBus(void);

           // Temporary  library development function  TODO: remove need for this

  void     pushSwapBytePixels(const void\* data\_in, uint32\_t len);

           // Same as setAddrWindow but exits with CGRAM in read mode

  void     readAddrWindow(int32\_t xs, int32\_t ys, int32\_t w, int32\_t h);

           // Byte read prototype

  uint8\_t  readByte(void);

           // GPIO parallel bus input/output direction control

  void     busDir(uint32\_t mask, uint8\_t mode);

           // Single GPIO input/output direction control

  void     gpioMode(uint8\_t gpio, uint8\_t mode);

           // Helper function: calculate distance of a point from a finite length line between two points

  float    wedgeLineDistance(float pax, float pay, float bax, float bay, float dr);

           // Display variant settings

  uint8\_t  tabcolor,                   // ST7735 screen protector "tab" colour (now invalid)

           colstart = 0, rowstart = 0; // Screen display area to CGRAM area coordinate offsets

           // Port and pin masks for control signals (ESP826 only) - TODO: remove need for this

  volatile uint32\_t \*dcport, \*csport;

  uint32\_t cspinmask, dcpinmask, wrpinmask, sclkpinmask;

           #if defined(ESP32\_PARALLEL)

           // Bit masks for ESP32 parallel bus interface

  uint32\_t xclr\_mask, xdir\_mask; // Port set/clear and direction control masks

           // Lookup table for ESP32 parallel bus interface uses 1kbyte RAM,

  uint32\_t xset\_mask[256]; // Makes Sprite rendering test 33% faster, for slower macro equivalent

                           // see commented out #define set\_mask(C) within TFT\_eSPI\_ESP32.h

           #endif

  //uint32\_t lastColor = 0xFFFF; // Last colour - used to minimise bit shifting overhead

  getColorCallback getColor = nullptr; // Smooth font callback function pointer

  bool     locked, inTransaction, lockTransaction; // SPI transaction and mutex lock flags

 //-------------------------------------- protected ----------------------------------//

 protected:

  //int32\_t  win\_xe, win\_ye;          // Window end coords - not needed

  int32\_t  \_init\_width, \_init\_height; // Display w/h as input, used by setRotation()

  int32\_t  \_width, \_height;           // Display w/h as modified by current rotation

  int32\_t  addr\_row, addr\_col;        // Window position - used to minimise window commands

  int16\_t  \_xPivot;   // TFT x pivot point coordinate for rotated Sprites

  int16\_t  \_yPivot;   // TFT x pivot point coordinate for rotated Sprites

  // Viewport variables

  int32\_t  \_vpX, \_vpY, \_vpW, \_vpH;    // Note: x start, y start, x end + 1, y end + 1

  int32\_t  \_xDatum;

  int32\_t  \_yDatum;

  int32\_t  \_xWidth;

  int32\_t  \_yHeight;

  bool     \_vpDatum;

  bool     \_vpOoB;

  int32\_t  cursor\_x, cursor\_y, padX;       // Text cursor x,y and padding setting

  int32\_t  bg\_cursor\_x;                    // Background fill cursor

  int32\_t  last\_cursor\_x;                  // Previous text cursor position when fill used

  uint32\_t fontsloaded;               // Bit field of fonts loaded

  uint8\_t  glyph\_ab,   // Smooth font glyph delta Y (height) above baseline

           glyph\_bb;   // Smooth font glyph delta Y (height) below baseline

  bool     isDigits;   // adjust bounding box for numbers to reduce visual jiggling

  bool     textwrapX, textwrapY;  // If set, 'wrap' text at right and optionally bottom edge of display

  bool     \_swapBytes; // Swap the byte order for TFT pushImage()

  bool     \_booted;    // init() or begin() has already run once

                       // User sketch manages these via set/getAttribute()

  bool     \_cp437;        // If set, use correct CP437 charset (default is ON)

  bool     \_utf8;         // If set, use UTF-8 decoder in print stream 'write()' function (default ON)

  bool     \_psram\_enable; // Enable PSRAM use for library functions (TBD) and Sprites

  uint32\_t \_lastColor; // Buffered value of last colour used

  bool     \_fillbg;    // Fill background flag (just for for smooth fonts at the moment)

#if defined (SSD1963\_DRIVER)

  uint16\_t Cswap;      // Swap buffer for SSD1963

  uint8\_t r6, g6, b6;  // RGB buffer for SSD1963

#endif

#ifdef LOAD\_GFXFF

  GFXfont  \*gfxFont;

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 9: TFT\_eSPI class conditional extensions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Load the Touch extension

#ifdef TOUCH\_CS

  #if defined (TFT\_PARALLEL\_8\_BIT) || defined (RP2040\_PIO\_INTERFACE)

    #if !defined(DISABLE\_ALL\_LIBRARY\_WARNINGS)

      #error >>>>------>> Touch functions not supported in 8/16 bit parallel mode or with RP2040 PIO.

    #endif

  #else

    #include "Extensions/Touch.h"        // Loaded if TOUCH\_CS is defined by user

  #endif

#else

    #if !defined(DISABLE\_ALL\_LIBRARY\_WARNINGS)

      #warning >>>>------>> TOUCH\_CS pin not defined, TFT\_eSPI touch functions will not be available!

    #endif

#endif

// Load the Anti-aliased font extension

#ifdef SMOOTH\_FONT

  #include "Extensions/Smooth\_font.h"  // Loaded if SMOOTH\_FONT is defined by user

#endif

}; // End of class TFT\_eSPI

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*                         Section 10: Additional extension classes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Load the Button Class

#include "Extensions/Button.h"

// Load the Sprite Class

#include "Extensions/Sprite.h"

#endif // ends #ifndef \_TFT\_eSPIH\_