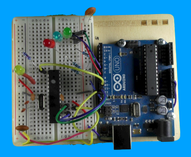
Eine eigene Klasse

Zur Displayansteuerung verwenden wir die Bibliothek TFT_eSPI (https://github.com/Bodmer/TFT_eSPI). Wir erstellen eine eigene Klasse, die alle Eigenschaften von TFT_eSPI erbt. In dieser Klasse werden wir Methoden erstellen, die unsere Bedürfnisse direkt unterstützen.

```
#include <TFT_eSPI.h>

class TFTEsp : public TFT_eSPI {
  public:
   TFTEsp();
    ...
  private:
     ...
};
```



Umgang mit Farben

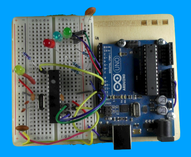
Wir definieren Standardfarben für den Vordergrund und den Hintergrund. Diese können mit den Funktionen **setFgColor(color)** und **setBgColor(color)** gesetzt werden. Die gesetzten Farben können auch wieder abgefragt werden (**getFgColor()**, **getBgColor()**).

Mit **cls()** (ClearScreen) wird der ganze Bildschirm mit der Hintergrundfarbe gefüllt.

Textausgaben erfolgen normalerweise in der Vordergrundfarbe mit einem Hintergrund in der Hintergrundfarbe.

```
// Bildschirm löschen
void cls();

// Farben
void setFgColor(uint16_t color);
void setBgColor(uint16_t color);
uint16_t getFgColor();
uint16_t getBgColor();
```



Ausnahmen in der Farbgebung

Die Standardfarben sind nicht in jedem Fall gültig.

Der Bildschirm kann durch **cls(color)** mit einer beliebigen Farbe gefüllt werden.

Für die Textausgabe können mit **setTextColor(fgColor, bgColor)** andere Farben gesetzt werden.

Die Standardfarben bleiben dabei unverändert.

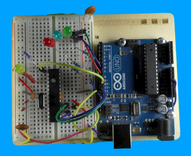
Mit **setTextColor()** wird die Textfarbe wieder auf die Standardfarben zurückgesetzt. Dies gilt ebenso, wenn **setFgColor** oder **setBgColor** aufgerufen wird.

```
// mit eigener Farbe löschen
void cls(uint16_t color);

// Mit Hintergrundfarbe löschen
void cls(); // mit Hintergrundfarbe löschen

// setzt eigene Textfarben
void setTextColor(uint16_t fgColor, uint16_t bgColor);

// setzt die Textfarbe wieder auf die Standardfarben
void setTextColor();
```



Das Koordinatensystem

Unser Display hat **320 x 240 Pixel.** Es kennt nur Grafikkordinaten und wir werden auch mit diesen arbeiten. Je nach Ausrichtung des Displays können es auch 240 x 320 Pixel sein.

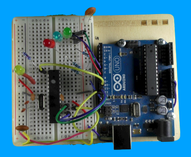
Ich arbeite im **Breitformat** mit den **Anschlüssen rechts**. Deshalb muss ich das Display mit **setRotation(1)** initialisieren.

Die Anzahl Pixel in x-Richtung können mit **width()** und die Anzahl Pixel in y-Richtung können mit **height()** abgefragt werden.

Die Koordinate 0,0 befindet sich links oben.

```
//  Diese Funktionen werden aus der
//  ursprünglichen Klasse übernommen
//  und müssen in unserer eigenen Klasse
//  nicht aufgeführt werden.

//  void setRotation(uint8_t rotation);
//  uint16_t width();
//  uint16_t height();
```



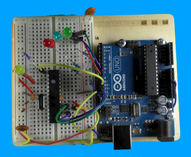
Einfache Textausgabe

Es stehen von der Basisklasse die Funktionen print und println zur Verfügung. Zur Positionierung wird setCursor verwendet.

Wir benötigen aber hauptsächlich eine Ausgabe an eine bestimmte Position. Dazu existiert bereits drawString(text,x,y).

drawString(text,x,y) schreibt den Text an die angegebene Stelle. Der dort bereits vorhandene Text wird gelöscht. Als Farbe werden die mit textColor gesetzten Werte verwendet.

```
// Übernahme aus ursprünglicher Klasse
// print(); println();
// setCursor(int16_t x, int16_t y);
// drawString(const char *string, int32_t x, int32_t y);
// drawString(const String& string, int32_t x, int32_t y);
```



Ausgabe in Felder

In Benutzeroberflächen benötigt man oft eine Ausgabe in Felder. Diese haben eine Breite, Höhe und Hintergrundfarbe.

Der Text hat eine Grösse, Farbe und eine bestimmte Ausrichtung im Feld.

Dazu definieren wir eine Feldklasse.

```
// Übernahme aus ursprünglicher Klasse
// print(); println();
// setCursor(int16_t x, int16_t y);
// drawString(const char *string, int32_t x, int32_t y);
// drawString(const String& string, int32_t x, int32_t y);
```