テキスト描画

テキスト表示

drawCenterString

関数プロトタイプ1:

```
void drawCenterString(const char *string, int32_t x, int32_t y, const IFont* font)
```

関数プロトタイプ2:

```
void drawCenterString(const char *string, int32_t x, int32_t y)
```

機能説明:

- 中央位置にテキストを描画します

引数:

- string：テキスト文字列
- x：描画開始 x 座標
- y：描画開始 y 座標

戻り値:

- null

drawChar

関数プロトタイプ1:

```
size_t drawChar(int32_t x, int32_t y, uint16_t uniCode, T color, T bg, float size_x, float size_y)
```

関数プロトタイプ2:

```
size_t drawChar(int32_t x, int32_t y, uint16_t uniCode, T color, T bg, float size)
```

関数プロトタイプ3:

```
size_t drawChar(uint16_t uniCode, int32_t x, int32_t y, uint8_t font)
```

関数プロトタイプ4:

```
size_t drawChar(uint16_t uniCode, int32_t x, int32_t y)
```

機能説明:

- 1文字を描画します

引数:

- x：描画開始 x 座標
- y：描画開始 y 座標
- uniCode：文字の UniCode コード
- color：文字色
- bg：文字背景色
- size(_N)：テキスト拡大率
- font：テキストフォント形式

戻り値:

- size_t:
  - 文字が x 方向に占める合計ピクセル幅

drawFloat

関数プロトタイプ1:

```
size_t drawFloat(float floatNumber, uint8_t dp, int32_t poX, int32_t poY)
```

関数プロトタイプ2:

```
size_t drawFloat(float floatNumber, uint8_t dp, int32_t poX, int32_t poY, uint8_t font)
```

機能説明:

- 浮動小数点数を描画します

引数:

- floatNumber：浮動小数点数
- dp：小数点以下の桁数
- poX：開始 x 座標
- poY：開始 y 座標
- font：テキストフォント形式

戻り値:

- size_t：文字列描画の合計水平ピクセル幅

drawNumber

関数プロトタイプ1:

```
size_t drawNumber(long long_num, int32_t poX, int32_t poY)
```

関数プロトタイプ2:

```
size_t drawNumber(long long_num, int32_t poX, int32_t poY, uint8_t font)
```

機能説明:

- long型の数値を描画します

引数:

- long_num：long型数値
- poX：x 座標
- poY：y 座標
- font：テキストフォント形式

戻り値:

- size_t：文字列描画の合計水平ピクセル幅

drawString

関数プロトタイプ1:

```
size_t drawString( const char *string, int32_t x, int32_t y)
```

関数プロトタイプ2:

```
size_t drawString( const char *string, int32_t x, int32_t y, const IFont* font)
```

機能説明:

- 文字列を描画します

引数:

- string：描画する文字列
- x：描画開始 x 座標
- y：描画開始 y 座標
- font：テキストフォント形式

戻り値:

- size_t：文字列描画の合計水平ピクセル幅

print

関数プロトタイプ1:

```
size_t print(char c)
size_t print(const char str[])
```

関数プロトタイプ2:

```
size_t print(int  n, int base = 10)
size_t print(long n, int base = 10)
size_t print(unsigned char n, int base = 10)
size_t print(unsigned int  n, int base = 10)
size_t print(unsigned long n, int base = 10)
size_t print(double        n, int digits= 2)
```

機能説明:

- カーソル位置にテキストを出力します。Arduino ヘッダーファイル `Print.h` と互換性があります。

引数:

- c：文字
- str：文字列
- n：整数または浮動小数点数
- base：基数（デフォルトは 10）
- digits：小数点以下の桁数（デフォルトは 2）

戻り値:

- size_t：出力した文字数

printf

関数プロトタイプ:

```
size_t printf(const char* format, ...)
```

機能説明:

- カーソル位置にテキストを出力します。Arduino ヘッダーファイル `LibPrint.h` と互換性があります。

引数:

- format：フォーマット文字列
- ...：可変引数

戻り値:

- size_t: 出力した文字数

println

関数プロトタイプ:

```
size_t println(void)
size_t println(char c)
size_t println(const char c[])
```

関数プロトタイプ:

```
size_t println(int  n, int base = 10)
size_t println(long n, int base = 10)
size_t println(unsigned char n, int base = 10)
size_t println(unsigned int  n, int base = 10)
size_t println(unsigned long n, int base = 10)
size_t println(double        n, int digits= 2)
```

機能説明:

- カーソル位置にテキストを出力します。Arduino ヘッダーファイル `Print.h` と互換性があります。

引数:

- c：文字/文字列
- n：整数または浮動小数点数
- base：基数（デフォルトは 10）
- digits：小数点以下の桁数（デフォルトは 2）

戻り値:

- size_t：出力した文字数

テキスト属性

setFont

関数プロトタイプ:

```
void setFont(const IFont* font)
```

機能説明:

- テキストフォントを設定します

引数:

- font：テキストフォント形式

戻り値:

- null

getFont

関数プロトタイプ:

```
const IFont* getFont (void)
```

機能説明:

- setFont で設定した現在使用中のフォントを取得します

引数:

- null

戻り値:

- const IFont*：現在使用中のフォントオブジェクトポインタ （Fontについて）

showFont

関数プロトタイプ:

```
void showFont(uint32_t td = 2000)
```

機能説明:

- 指定した時間テキストを表示します

引数:

- td：表示時間（ミリ秒単位）

戻り値:

- null

fontHeight

関数プロトタイプ:

```
int32_t fontHeight(const IFont* font)
```

機能説明:

- 指定したフォントの高さを取得します

引数:

- font：テキストフォント形式

戻り値:

- int32_t：フォントの高さ

fontWidth

関数プロトタイプ:

```
int32_t fontWidth(const IFont* font)
```

機能説明:

- 指定したフォントの幅を取得します

引数:

- font：テキストフォント形式

戻り値:

- int32_t：フォントの幅

loadFont

関数プロトタイプ1:

```
bool loadFont(const uint8_t* array)
```

関数プロトタイプ2:

```
bool loadFont(T &fs, const char *path)
```

関数プロトタイプ3:

```
bool loadFont(const char *path)
```

関数プロトタイプ4:

```
bool loadFont(DataWrapper* data)
```

機能説明:

- フォントデータを読み込みます

*この関数の使い方はこちらを参照してください

引数:

- array：フォントデータへのポインタ
- fs：フォントデータオブジェクト
  - SPIFFS
  - SD など
- path：フォントファイルパス
- data：フォントデータオブジェクトへのポインタ

戻り値:

- bool：読み込みの成否
  - true：成功
  - false：失敗

unloadFont

関数プロトタイプ:

```
void unloadFont(void)
```

機能説明:

- setFont で設定したフォントをデフォルトフォント `&fonts::Font0` に戻します

引数:

- null

戻り値:

- null

setTextColor

関数プロトタイプ1:

```
void setTextColor(T color)
```

関数プロトタイプ2:

```
void setTextColor(T1 fgcolor, T2 bgcolor)
```

機能説明:

- テキスト色を設定します

引数:

- color：テキスト色
- fgcolor：テキスト前景色
- bgcolor：テキスト背景色

戻り値:

- null

setTextSize

関数プロトタイプ1:

```
void setTextSize(float size)
```

関数プロトタイプ2:

```
void setTextSize(float sx, float sy)
```

機能説明:

- テキストサイズを設定します

引数:

- size：テキスト拡大率
- sx：テキスト x 軸方向拡大率
- sy：テキスト y 軸方向拡大率

戻り値:

- null

getTextSizeX

関数プロトタイプ:

```
float getTextSizeX(void)
```

機能説明:

- setTextSize で設定したテキスト幅を取得します

引数:

- null

戻り値:

- float：テキスト幅

getTextSizeY

関数プロトタイプ:

```
float getTextSizeY(void)
```

機能説明:

- setTextSize で設定したテキスト高さを取得します

引数:

- null

戻り値:

- float：テキスト高さ

setTextStyle

関数プロトタイプ:

```
void setTextStyle(const TextStyle& text_style)
```

機能説明:

- テキストスタイルを設定します

引数:

- text_style：テキストスタイルオブジェクト （TextStyleについて）

戻り値:

- null

getTextStyle

関数プロトタイプ:

```
TextStyle& getTextStyle(void)
```

機能説明:

- setTextStyle で設定したテキストスタイル情報を取得します

引数:

- null

戻り値:

- TextStyle&: テキストスタイル情報の参照

textLength

関数プロトタイプ:

```
int32_t textLength(const char *string, int32_t width)
```

機能説明:

- 指定範囲内で表示できる文字数を取得します

引数:

- string: 表示する文字列へのポインタ
- width: 指定範囲のピクセル幅

戻り値:

- int32_t: 表示できるバイト数

> **特別注意:**
>
> 戻り値はバイト数であり文字数ではありません。日本語などマルチバイト文字の場合、バイト数は文字数より多くなります。

textWidth

関数プロトタイプ:

```
int32_t textWidth(const char *string, const IFont* font)
```

機能説明:

- 画面上に表示される文字列の幅を返します。フォントを指定しない場合はデフォルトフォントまたは setFont で設定したフォントで計算します。

引数:

- string: 画面に表示する文字列へのポインタ
- font: テキストフォント形式

戻り値:

- int32_t：表示される文字列のピクセル幅

テキストの配置とマージン

setTextDatum

関数プロトタイプ:

```
void setTextDatum(textdatum_t datum)
```

機能説明:

- テキストの基準情報を設定します

引数:

- datum：テキスト整列方式

> **説明:**
>
> この整列方式は、指定座標点がテキスト内容に対してどの位置になるかを示します。詳細な整列方式は `textdatum_t` を参照してください。

戻り値:

- null

getTextDatum

関数プロトタイプ:

```
textdatum_t getTextDatum(void)
```

機能説明:

- setTextDatum で設定したテキスト基準情報を取得します

引数:

- null

戻り値:

- textdatum_t：テキスト整列方式

setTextPadding

関数プロトタイプ:

```
void setTextPadding(uint32_t padding_x)
```

機能説明:

- テキストのパディング値を設定します

引数:

- padding_x：テキストパディング値（ピクセル単位）

戻り値:

- null

getTextPadding

関数プロトタイプ:

```
uint32_t getTextPadding(void)
```

機能説明:

- setTextPadding で設定したテキストパディング値を取得します

引数:

- null

戻り値:

- uint32_t：テキストパディング値

自動改行

setTextWrap

関数プロトタイプ:

```
void setTextWrap( bool wrapX, bool wrapY = false)
```

機能説明:

- テキストの自動改行を設定します

引数:

- wrapX：テキスト x 軸方向自動改行フラグ
- wrapY：テキスト y 軸方向自動改行フラグ（デフォルトは改行しない）

戻り値:

- null

> **説明:**
>
> この関数を使用しない場合、システムはデフォルトでx軸方向の自動改行を有効にします。

サンプルプログラム:

cpp

```cpp
#include <Arduino.h>
#include <M5GFX.h>

M5GFX display;

void setup() {
    display.begin();

    display.setRotation(3);
    if(display.isEPD())
    {
        display.setColorDepth(8);//電子ペーパー製品は最大8ビットのビット深度をサポートします。
        display.setEpdMode(epd_fastest);
    }
    else
    {
        display.setColorDepth(16);
    }

    display.clear(TFT_WHITE);
    display.setFont(&fonts::FreeMonoBoldOblique24pt7b);
    display.setTextColor(TFT_BLACK);
    display.setTextWrap(true);
    display.setCursor(0, 0);
    display.setTextSize(1,2);
    uint16_t x = display.getTextSizeX();
    uint16_t y = display.getTextSizeY();
    display.printf("Text SizeX:%d SizeY:%d", x, y);
    display.setTextSize(1,1);
    display.setTextDatum(top_left);//この整列方式は、指定座標点がテキスト内容に対してどの位置になるかを示します。
    x = display.fontWidth(&fonts::Font0);
    y = display.fontHeight(&fonts::Font0);
    display.printf(" Font0 W:%d H:%d", x, y);
    display.setTextPadding(10);
    x = display.width() / 2;
    y = display.height() / 2;

    display.drawCenterString("This is CenterString", x, y);
    display.drawChar(0x004D, 0, y/4*7-24);//M
    display.drawChar(0x0035, 30, y/4*7-24);//5
    display.drawChar(0x0053, 30*2, y/4*7-24);//S
    display.drawChar(0x0074, 30*3, y/4*7-24);//t
    display.drawChar(0x0061, 30*4, y/4*7-24);//a
    display.drawChar(0x0063, 30*5, y/4*7-24);//c
    display.drawChar(0x006B, 30*6, y/4*7-24);//k
    display.drawFloat(127.45678, 5, 0, y/4*7);
    display.drawNumber(1234567890, 0, y*2-40);
    display.drawString("This is drawString", x/2, y/4*7);
}

```

```
51 void loop() {
52 }
```