PocketStar Library Documentation

```
isSDConnected()
```

Introduction

New Sketch & Basic Functions

Creating a new sketch in Arduino with the default File > New results in an incopatible sketch; For the PocketStar, you have to go to *File* > *Examples* > *PocketStar Library* > *NewSketch.*

Every PocketStar game must declare four functions:

`void initialize()`
This function is called once after starting. Code which should be executed once, like loading previous high scores from the SD, should be placed here.

`void update(float tpf)`
This gets called continuously; put the main code in here. The `tpf` variable contains the time (in seconds) which has passed since the last time this function got called.

`boolean pause()`
Called before entering pause menu. You may want to save the progress of the game here, since the user may exit the game.
Return `false` to prevent showing the pause menu; you should not always prevent it, though, since the user won't be able to go back to the main menu otherwise.

`void resume()`
Called after leaving the pause menu. Redraw the entire screen here to ensure the pause menu is no longer visible.

Uploading

Three different types are aviable at *Tools* > *Build Type* to fit different scenarios:

Default	To upload the code directly to the PocketStar, keeping the USB connection.
Menu	To upload the code to the PocketStar, disabling its USB connection. This prevents crashes while loading games. It won't be recognized by the PC anymore; you have to set it into Bootloader mode first.
Game	To export the code to a game binary; see below for more details.

Select the PocketStar at *Tools* > Port (or Arduino / Genuino Zero while in Bootloader mode) before uploading.

Exporting as Game

To export the sketch as game, select at *Tools* > *Build Type* > *Game*, then *Sketch* > *Export compiled Binary.* Locate the binary inside the folder of your sketch, it should be named *[sketchname].ino.pocketstar.bin*. Rename it to a fitting name, e.g. *pocketris.bin*, and copy it.
Connect the SD card to the PC and open the *games* folder in it. Create a new folder named exacly as the binary and paste the binary into it. If you have a thumbnail for your game (a .psi file created with the Image Converter, see below), you have to put it into this folder, too, also names exactly as the binary. The folder structure has to look like this:

```
[SD Root]
  └─games
      └─yourgame
          ├─yourgame.bin
          └─yourgame.psi
```

Apps have the same file structure, but inside the *apps* folder instead.

Reuploading the Main Menu

The menu can be found at *File* > *Examples* > *PocketStar Library* > *MainMenu.* Make sure *Tools* > *Build Type* > *Menu* is selected before you upload it; otherwise, the PocketStar may crash while loading games.

Image Converter

An image converter is included in the library inside the *extra* folder. It needs Java 8 or newer to run.
Press 'Choose image...' and select the image you want to convert. Only png and jpg files can be converted; if your image has a different type, open it in an image editor and save it as png or jpg (png is recommended since it is a lossless format). You can choose several settings before converting:

Type:
'convert to code' creates a txt file with code which can be pasted into sketches. Use the library to draw them.
'convert to file' creates a file with 'psi' ending (PocketStar Image); this is used to draw the thumbnails of games and apps in the menu.

Bit depth:
8 bits (RGB332) or 16 bits (RGB565) per pixel. The The `drawImage` function in the library supports both bit dephts, `drawSprites` only supports 8 bit.

1- or 2-dimensional:
Only for conversion into code. Determines if the pixel data is stored in a one- or twodimensional array. The PocketStar library uses onedimensional arrays.

Alpha:
If checked, the converter will mark transparent pixels a special value. The `drawSprites` funcion in the library will skip these pixels. The `drawImage` fucntion does not support alpha; using it may result in strangely colored pixels.
Semi-transparent pixels are not supported, and will be converted to fully opaque pixels if their alpha value is greater than 128, ot to transparent pixels otherwise.

Converting creates a new file named like the image, but with a txt or psi extension. The txt files contain the converted code; the image variable will have the same name as the file.

Constants

Color constants

BLACK_16b
DARKGRAY_16b
GRAY_16b
LIGHTGRAY_16b
WHITE_16b
RED_16b
ORANGE_16b
YELLOW_16b
GREENYELLOW_16b
GREEN_16b
CYAN_16b
BLUE_16b
PURPLE_16b
MAGENTA_16b
DARKRED_16b
DARKGREEN_16b
DARKBLUE_16b

BLACK_8b
DARKGRAY_8b
GRAY_8b
LIGHTGRAY_8b
WHITE_8b
RED_8b
ORANGE_8b
YELLOW_8b
GREENYELLOW_8b
GREEN_8b
CYAN_8b
BLUE_8b
PURPLE_8b
MAGENTA_8b
DARKRED_8b
DARKGREEN_8b
DARKBLUE_8b

Bitdepth constants

BitDepth8
BitDepth16

Color Modes

ColorModeRGB
ColorModeBGR

Buttons

ButtonUp
ButtonDown
ButtonLeft
ButtonRight
ButtonA
ButtonB

Backlight Colors

BacklightBlack
BacklightRed
BacklightGreen
BacklightBlue
BacklightYellow
BacklightMagenta
BacklightCyan
BacklightWhite

Fonts

pocketStar5pt
pocketStar6pt
pocketStar7pt
pocketStar12pt
pocketStar16pt
pocketStar26pt
pocketStarSymbols

The 5pt, 6pt and 7pt fonts contain these characters:
A-Z a-z 0-9 .,"'?!@_*#$%&()+-/:;<=>[\]^`{|}~

The 12pt, 16pt and 26pt fonts contain these characters:

0-9 .,"':

The symbols font contains these characters:

char	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
symbol	←	→	↑	↓	↖	↗	↙	↘	⇔	⇕	⇐	⇒	★	♥	☺	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	☺	[illegible]
char	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
symbol	⏵	⏸	■	◀	▶	⏮	⏭	⏩	⏪	↰	≤	≥	±	α	β	λ	π	Ω	©	®	™	€	■	✚	▲	●
char	0	1	2	3	4	5	6	7	8	9																
symbol	■	⚀	⚁	⚂	⚃	⚄	⚅	[illegible]	[illegible]	[illegible]																

To print a symbol, you have to print the char which represents it.

General Control

```
void setBitDepth(uint8_t bitDepth)
void setColorMode(uint8_t colorMode)
void setMirror(boolean mirror)
```

Description

Used to change basic screen properties.

setMirror

Mirrors the screen across the Y-axis.

setBitDepth

Changes the bit depth, and therefore determines the amount of possible colors. Aviable modes are `BitDepth8` (256 colors) and `BitDepth16` (65k colors).

setColorMode

Changes the color mode, aviable are `ColorModeRGB` and `ColorModeBGR`.

Parameters

`bitDepth`	either `BitDepth8` or `BitDepth16`
`colorMode`	either `ColorModeRGB` or `ColorModeBGR`
`mirror`	`true` to mirror future drawing calls

Accelerated Graphics

```
void clearWindow(uint8_t x, uint8_t y, uint8_t width, uint8_t height)
void clearScreen()
```

Description

Clears the specified are of the screen, or the entire screen.

Parameters

`x, y`	the origin coordinates
`colorMode`	the bounds of the window

```
void drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t color)
void drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint16_t color)
void drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t red,
      uint8_t green, uint8_t blue)
```

Description

Draws a line from (x1, y1) to (x2, y2) in the specified color. These functions are independent of the current bit depth.

Parameters

`x1, y1`	the coordinates of the start of the line
`x2, y2`	the coordinates of the start of the line
`color`	the color of the line
`red, green, blue`	the color of the line, stored in the least six bits

See also

`drawRect()`
Color constants

```
void drawRect(uint8_t x, uint8_t y, uint8_t width, uint8_t height, boolean fill,
      uint8_t color)
void drawRect(uint8_t x, uint8_t y, uint8_t width, uint8_t height, boolean fill,
      uint16_t color)
void drawRect(uint8_t x, uint8_t y, uint8_t width, uint8_t height, boolean fill,
      uint8_t red, uint8_t green, uint8_t blue)
```

Description

Draws a rectangle from (x, y) with the specified width and height in the specified color. These functions are independent of the current bit depth.

Parameters

`x, y`	the origin coordinates (upper left)
`width, height`	the bounds of the rectangle
`fill`	wether the rectangle should be filled or only outline
`color`	the color of the line
`red, green, blue`	the color of the line, stored in the least six bits

See also

`drawLine()`
Color constants

Pixel Manipulation

```
void drawPixel(uint8_t x, uint8_t y, uint16_t color)
```

Description
Changes a single pixel to the specified color.

Parameters
`x, y`	coordinates of the pixel
`color`	the new color

See also
Color constants

```
void setX(uint8_t start, uint8_t end)
void setY(uint8_t start, uint8_t end)
void goTo(uint8_t x, uint8_t y)
```

Description

Used to set the bounds for a following call to writeBuffer().
`goTo(x, y)` does the same as `setX(x, 95)`, `setY(y, 63)`.
Usually, you first set the drawing boundaries with `setX()` and `setY()`, then use `startTransfer()` followed by one or more calls to `writeBuffer()`, and finish the transfer with `endTransfer()`.

Parameters

`start, end`	the row / column bounds
`x, y`	the start coordinates

See also

Color constants

```
void startTransfer()
void endTransfer()
```

Description

Used to set up a data transfer to the screen.
Usually, you first set the drawing boundaries with `setX()` and `setY()`, then use `startTransfer()` followed by one or more calls to `writeBuffer()`, and finish the transfer with `endTransfer()`.

See also

`setX()`
`setY()`
`writeBuffer()`

```
void writeBuffer(const uint8_t *buffer, int count)
void writeBuffer(const uint16_t *buffer, int count)
```

Description

Sends the colors to the screen, row by row.
You cannot change the bit depth after starting a transfer; thus, all colors used here should have the same bit depth.
Usually, you first set the drawing boundaries with `setX()` and `setY()`, then use `startTransfer()` followed by one or more calls to `writeBuffer()`, and finish the transfer with `endTransfer()`.

Parameters

`buffer`	an array of colours
`count`	length of the array

See also

`void startTransfer()`
`void endTransfer()`
`setX()`
`setY()`

Images and Sprites

```
void drawImage(uint8_t x, uint8_t y, const PSImage *image)
```

Description

Draws an Image on the screen. The upper left corner is located at the given x/y position. Use the Image Converter to convert images to code.
This method does **not** check if the correct bit depth is set; ensure this before calling `drawImage()`.

Parameters

`x, y`	the coordinate of the upper left corner
`image`	length of the array a pointer to a PSImage

See also

Image Converter

```
void drawSprites(const PSSprite *sprites[], uint8_t numSprites)
void drawSprites(const PSSprite *sprites[], uint8_t numSprites, uint8_t background)
void drawSprites(const PSSprite *sprites[], uint8_t numSprites, uint8_t background,
      uint8_t ySkip)
void drawSprites(const PSSprite *sprites[], uint8_t numSprites,
      const PSImage *background)
void drawSprites(const PSSprite *sprites[], uint8_t numSprites,
      const PSImage *background, uint8_t ySkip)
```

Description

Draws the given sprites on the screen, with the last sprite in the array being displayed above the others.
A PSSprite consists of:

- `const PSImage *image` the Image (bit depth must be 8); can contain transparency
- `short x, y` coordinates (upper left) of the sprite
- `boolean enabled` determines if the sprite should be drawn
- `boolean mirrorFlag` mirrors the sprite

Use the Image Converter to convert sprites to code.
Sprites can be partially out of the screen bounds. If ySkip is > 0, the first rows of the top of the screen will be skipped. This space can be used to display text (e.g. a score).

Parameters

`sprites`	an array of pointers to sprites
`numSprites`	length of the array
`background`	length of the array a background color or image; black if none is given
`ySkip`	the number of rows to skip at the top

See also

Color constants
Image Converter

Printing

```
void setFont(const PSFont &font)
```

Description
Sets the font used for future printing.

Parameters
`font`	the font

See also
Fonts
`setFontColor()`
`print()`

```
void setFontColor(uint16_t color, uint16_t colorBackground)
```

Description
Changes the text color.

Parameters
`color`	the text color
`colorbackground`	the color used for pixels not covered by text

See also
Color constants
`setFont()`
`print()`

```
void setCursor(uint8_t x, uint8_t y)
```

Description

Sets the cursor position; this will be the upper-left corner of text printed afterwards.

Parameters

`x, y`	the cursor position

See also

`print()`

```
void print(...)
```

Description
Prints the given data; integers and floats will be converted to text automatically. This function advances the cursor position automatically, further printing will start at the end of the given text.

Parameters
A string, integer or float you want to print.

See also
`setFont()`
`setFontColor()`
`getPrintWidth()`
`print()`

```
uint8_t getPrintWidth(char *text)
```

Description

Returns the width the given text would have on the display using the current font. This can be used to align text.
Unlike print(), it does not accept integers or floats, so convert them manually before calling this function.

Parameters

`text`	the text

See also

`setFont()`
`print()`

Buttons, Vibration and Backlight

```
uint8_t getButtons()
uint8_t getButtons(uint8_t buttonMask)
```

Description
Returns which buttons are pressed, with an optional button mask. This does not cover the pause button.

Parameters
`buttonMask`	a mask of one or more buttons

See also
Buttons

```
void vibrate(boolean vibrate)
```

Description

Starts or stops the vibration motor.
Do not use this constantly, as it causes the motor to heat up. Exaggerated use may damage the device.

Parameters

`vibrate`	`true` to vibrate, `false` to stop

```
void setBacklightColor(uint8_t color)
void disablebacklight()
```

Description
Sets the color of the backlight LED or turns it off.

Parameters
`color`	one of the backlight color constants

See also
Backlight colors

Battery and SD Card Information

```
float batteryLevel()
```

Description

Returns the current battery percentage left. This is not completely accurate and may differ from the actual battery level, up to about 5%, but is guaranteed to be between 0 and 1. While charging, the value returned by this function can be much higher than expected, up to 30%.

See also

`isCharging()`

```
boolean isCharging()
```

Description

Returns `true` if the PocketStar is charging, `false` otherwise.

See also

`batteryLevel()`

```
boolean isSDConnected()
```

Description

Returns `true` if an SD card is inside the SD slot, `false` otherwise.