PinLightShield Library Documentation

Version 0.1

19.03.15

Author: Herbert Koller

1 Table of Contents

2 Introduction

This library was implemented to make the use of the PinLightShield easier for people who are not so familiar with writing code. It implements most basic functions that will be needed over and over when you try to build your own pin lightshow using the PinLightShield or your own Arduino based hardware.

There will still be some coding necessary to implement the particular logic for **your** pinball machine. Imagine you want to put a Shaker into your pin and in the first step the Shaker shall be activated when the pin starts a certain mode (e.g. multiball or whatever). Then you need to find the criteria that tell you that your pin just entered that mode. In the case of Multiball this might be the number of balls that are still in the trough. This can be determined by reading some of the trough Switches. For other modes you might need to determine the state of one or more Inserts or Switches.

The logic that analyzes the trough switches (like “if trough1 and trough2 are empty then activate Shaker”) you will still have to write yourself, because that is specific to the pin and what you want to achieve. This library however will relieve you from writing those low level functions that read Switches, get the state of Inserts, activate RGB-Strips or Shakers and so on.

This library has been tested thoroughly on several pinball machines (WPC, Stern, DE) but it still may cause issues under certain circumstances and therefore the author of this library takes no responsibility whatsoever for eventual problems or even damage to your machine (which is kind of hard to imagine anyway).

There might also be functions you´re missing. Please let me know if such issues arise and I will try to fix them as quickly as possible.

3 Installation of the Library

For installation just extract the complete PLS directory into the Arduino libraries directory, which usually is My Documents/Arduino/libraries. Then launch your Arduino IDE (Integrated Development Environment) and go to “Sketch → Import Library”. You should see PLS inside this menu.

Execute that command for every sketch that wants to make use of this library. The command will insert the line

```
#include <pls.h>
```

into your sketch. The library will be compiled together with your sketch.

4 Contents of the Library

This library provides a number of classes that can be used with the PinLightShield, but also with your own hardware if it´s based on the same concepts as the PinLightShield. Those classes and their functions will be described in the following sections.

4.1 Useful Utility Functions

4.1.1 CopyColor

Copies RGB colors from one variable to another.

Exampe:

```
int yellow[3] = { 102, 242, 0 };

int destColor[3];

CopyColor(destColor, yellow);    // sets destColor to yellow
```

4.1.2 RGB2Long

Converts an RGB color into a packed format

Exampe:

```
unsigned long destColor;

destColor  = RGB2Long(0xFF, 0x00, 0x00);    // sets destColor to red
```

4.1.3 Long2RGB

Converts packed color into individual values for red, green and blue or a color array. Can be called in 3 ways.

Exampe:

```
#define MYBLUE                    0x191970

byte red, green, blue;
int redint, greenint, blueint;
int RGB[3];

Long2RGB(MYBLUE, &red, &green, &blue);    // red = 0x19, green = 0x19, blue = 0x70
Long2RGB(MYBLUE, &redint, &greenint, &blueint);    // redint = 0x19, greenint = 0x19, blueint = 0x70
Long2RGB(MYBLUE, RGB);    // RGB[0] = 0x19,  RGB[1] = 0x19, RGB[2] = 0x70
```

4.1.4 GetRed, GetGreen, GetBlue

Get the value of the individual components from a packed color.

Exampe:

```
#define MIDNIGHTBLUE              0x191970

byte red;
byte green;
byte blue;
```

```
red = GetRed(MIDNIGHTBLUE);         // red = 0x19
green = GetGreen(MIDNIGHTBLUE);     // green = 0x19
blue = GetBlue(MIDNIGHTBLUE);       // blue = 0x70
```

4.2 *Class RGBStrip*

This class provides useful functions to control an RGB LED or an RGB LED strip.

4.2.1 Instantiation

For every device you want to control from the PinLightShield you have to first call its instantiator. That call has to happen before the setup() function in your sketch.

On creation this class takes 4 parameters, the last one being *optional*. The instantiation has always to happen before the setup() function of your sketch.

1. int redpin: the PinLightShield pin that connects to the “red” signal of the strip
2. int greenpin: the PinLightShield pin that connects to the “green” signal of the strip
3. int bluepin: the PinLightShield pin that connects to the “blue” signal of the strip
4. *int brightness: the brightness of the RGB strip (1 – 100); defaults to 100 if not specified*

Example:

```
RGBStrip SurroundStrip(5, 6, 3, 80);    // this creates a strip called SurroundStrip on pins 3, 5 and 6
.                                       // of the PinLightShield and sets the brightness to 80%
.
.
void setup() {
}
```

4.2.2 Function LightStrip

This function lights the strip in a specific color and you can pass the color in 3 different forms.

1. three individual values of type int
2. an array[3] of type int
3. a packed color value of type unsigned long

Example:

```
#define MIDNIGHTBLUE        0x191970
int yellow[3] = { 102, 242, 0 };
```

```
SurroundStrip.LightStrip(255, 255, 255);        // lights the strip full white, if “brightness” is at 100%
SurroundStrip.LightStrip(yellow);
SurroundStrip.LightStrip(MIDNIGHTBLUE);
```

4.2.3 Function SwitchOff

Switches the RGB Strip off.

Example:

```
SurroundStrip.SwitchOff();
```

4.2.4 Function SetBrightness

This function takes one parameter and sets the brightness to a value between 0 and 100. In the current implementation of this library the “brightness” affects everything you do with the strip, no matter which effect you´re using.

1.	int brightness:	the brightness of the RGB strip (1 – 100)

Example:

```
SurroundStrip.SetBrightness(50);                // sets the brightness of the strip to 50%
```

4.2.5 Function MakeFlashes

This function causes the strip to flash a certain number of times in a certain color with a certain length of flashes. It takes 3 parameters.

1.	int color[3]:	an array of 3 containing the values for red, green and blue
2.	int flashes:	the number of flashes
3.	int flashlength:	the length of each flash in ms

Example:

```
int yellow[3] = { 102, 242, 0 };
.
.
.
void loop() {
.
.
.
SurroundStrip.MakeFlashes(yellow, 5, 100);      // flashes yellow 5 times for each 100ms
.
.
}
```

NOTE: Use this function very carefully as it uses the delay() function to control the flashlength meaning that nothing else will happen while this function is executed.

4.2.6 Function RainbowColorChange

This function executes a rainbow-like color effect on the RGB strip. It takes the current time in ms as the only argument, but also depends on the class internal variable _fadespeed_rainbow, which gets a default value (7) during the creation of the class, but can be changed with the function SetRainbowSpeed (see below).

1. unsigned long CurrentMillis: the current time in ms

Example:

```
unsigned long CurrentMillis;      // global variable stores the current time in ms
.
void setup() {
.
.
SurroundStrip.SetRainbowSpeed(3);      // causes the rainbow to change colors rather slowly
.
}

void loop() {
CurrentMillis = millis();              // store the current time in ms
.
SurroundStrip.RainbowColorChange(CurrentMillis);
.
.
}
```

4.2.7 Function SetRainbowSpeed

This function sets the speed of the rainbow color change.

1. int rainbowspeed: determines the speed of the color change (higher = slower)

Example:

```
SurroundStrip.SetRainbowSpeed(7);      // the color change will happen rather slowly
```

4.2.8 Function SetupMultiColorFlash

This function prepares the strip for a sequence of flashes of up to 5 different colors. The length of each flash can be defined individually and the total length of the whole MultiColorFlash can be defined as well. Normally the color flashes will occur in the sequence in which they are defined. With the parameter "randsequence" they can occur in a random sequence. For that purpose you should call the Arduino function randomSeed() somewhere earlier in your sketch (most likely in setup().

1. byte nrofcolors: determines the number of colors you want to use (1 – 5)
2. unsigned long colors[5]: the array containing up to five colors
3. int durations[5]: the array containing the durations of each color
4. Boolean randsequence: determines if the flashes occur in sequence or randomly
5. int FlashDuration: the total minimum length of the effect

4.2.9 Function MultiColorFlash

This function causes the LED to display a rapid succession of flashes in up to five different colors. The colors, the length of the flashes and the total duration of the flash can be defined with the function SetupMultiColorFlash (see above). MultiColorFlash itself only takes two parameters.

1. unsigned long CurrentMillis: the current time in ms (must be determined before the call to this function by calling the Arduino function millis().
2. FlashActive: this parameter is used as an input and is written as an output, therefore it must be called with an & in front. This variable is used to make sure that the flash lasts for a minimum period of time. [1]

Example:

```
#define TEAL                0x008080
#define LIME                0x00FF00
#define SPRINGGREEN         0x00FF7F
#define AQUA                0x00FFFF
#define CYAN                0x00FFFF

unsigned long CurrentMillis;    // global variable stores the current time in ms
boolean FlashActive;
unsigned long colors[5] = {TEAL, LIME, SPRINGGREEN, AQUA, CYAN};
int durations[5] = {200, 100, 200, 100, 50};  // e.g. TEAL will always last 200ms, LIME only 100
RGBStrip SurroundStrip(5, 6, 3);            // instantiates the RGB strip
```

[1] The signal of a coil or flasher typically only last 15-30 ms, which would make the flash hardly noticeable. Therefore this offers the chance to make the flash last as long as you want as long as no other function interferes with it.

```
.
void setup() {
.
randomSeed();               // if you want to use a random sequence of flashes
}

void loop() {
CurrentMillis = millis();               // store the current time in ms
.
.
if (CannonCoil.ReadInput() || FlashActive)  // if Cannon coil active and mimimum duration not over
  {
  if (!FlashActive)    // start Cannon Flash Mode
    {
    SurroundStrip.SetupMultiColorFlash(5, colors, durations, false, 2000);  // no random, 2000ms total
    }
  SurroundStrip.MultiColorFlash(CurrentMillis, &FlashActive);  // function sets FlashActive
  }
.
.
}
```

In this example the LED strip will flash orange and white for 500ms, when the orange flash will last 100ms and the white flash 50ms. Notice the “&” before “FlashActive”. That´s important!!!

4.2.10 Function SetupTwoColorFlash

This function prepares the RGB-Strip to execute a flash of two alternating colors. It defines the colors to use, the length of each flash and the total minimum duration of this effect.

1. int color1[3]:	the first color
2. int color2[3]:	the second color
3. int Col1Duration:	the length of the 1st color flash in ms
4. int Col2Duration:	the length of the 2nd color flash in ms
5. int FlashDuration:	the total minimum length of the effect

This function must be called before you can call the function TwoColorFlash that actually executes the effect. It must be called only once, though, as long as you don´t want to change colors or durations etc.

You´ll find an example how to use this function down below where TwoColorFlash is explained.

4.2.11 Function TwoColorFlash

This function causes the LED display a rapid succession of flashes in two different colors. The colors, the length of the flashes and the total duration of the flash can be defined with the function SetupTwoColorFlash (see above). TwoColorFlash itself only takes two parameters.

3. unsigned long CurrentMillis: the current time in ms (must be determined before the call to this function by calling the Arduino function millis().
4. FlashActive: this parameter is used as an input and is written as an output, therefore it must be called with an & in front. This variable is used to make sure that the flash lasts for a minimum period of time. [2]

Example:

```
unsigned long CurrentMillis;       // global variable stores the current time in ms
boolean FlashActive;
int white[3]  = { 255, 255, 255 };
int orange[3] = { 255, 100, 0 };
RGBStrip SurroundStrip(5, 6, 3);             // instantiates the RGB strip
.
void setup() {
.
.
}

void loop() {
CurrentMillis = millis();               // store the current time in ms
.
.
if (CannonCoil.ReadInput() || FlashActive)  // if Cannon coil active and mimimum duration not over
  {
  if (!FlashActive)    // start Cannon Flash Mode
    {
    SurroundStrip.SetupTwoColorFlash(orange, white, 100, 50, 500);
    }
  SurroundStrip.TwoColorFlash(CurrentMillis, &FlashActive);  // function sets FlashActive
  }
.
.
}
```

[2] The signal of a coil or flasher typically only last 15-30 ms, which would make the flash hardly noticeable. Therefore this offers the chance to make the flash last as long as you want as long as no other function interferes with it.

In this example the LED strip will flash orange and white for 500ms, when the orange flash will last 100ms and the white flash 50ms. Notice the "&" before "FlashActive". That´s important!!!

4.2.12 Function SetupTwoColorFade

This function prepares the RGB-Strip for a fading effect that slowly changes the light from one to color to another one (and back of course). With this function you define the two colors, the speed of the fading effect and the total duration of it.

1. unsigned long fadeColorFrom: the color to start from
2. unsigned long fadeColorTo: the color to fade to
3. int fadestep: the size of the steps the colors take in each iteration (1 - 5)
4. int fadespeed: the speed determines how quickly the fade will happen
5. int FadeDuration: the total minimum length of the effect

This function must be called before you can call the function TwoColorFade that actually executes the effect. It must be called only once, though, as long as you don´t want to change colors or durations etc.

You´ll find an example how to use this function down below where TwoColorFade is explained.

4.2.13 Function TwoColorFade

This function fades the RGB strip back and forth between two colors. The colors, the speed and the minimum duration of the effect can be defined individually with the function SetupTwoColorFade (see above). TwoColorFade itself only takes two parameters.

.

1. unsigned long CurrentMillis: the current time in ms (must be determined before the call to this function by calling the Arduino function millis().
2. FadeActive: this parameter is used as an input and is written as an output, therefore it must be called with an & in front. This variable is used to make sure that the fade lasts for a minimum period of time. [3]

Example:

```
unsigned long CurrentMillis;      // global variable stores the current time in ms
Boolean FadeActive;
#define WHITE                     0xFFFFFF
```

[3] The signal of a coil or flasher typically only last 15-30 ms, which would make the flash hardly noticeable. Therefore this offers the chance to make the fade last as long as you want as long as no other function interferes with it.

```
#define ORANGERED                    0xFF4500
RGBStrip SurroundStrip(5, 6, 3);               // instantiates the RGB strip
.
void setup() {
.
.
}

void loop() {
CurrentMillis = millis();                // store the current time in ms
.
.
if (CannonCoil.ReadInput() || FadeActive)  // if Cannon coil active and mimimum duration not over
  {
  if (!FadeActive)    // start Fade Mode
    {
    SurroundStrip.SetupTwoColorFade(ORANGERED, WHITE, 1, 1, 500);
    }
  SurroundStrip.TwoColorFade(CurrentMillis, &FadeActive);  // function sets CannonActive
  }
.
.
}
```

In this example the LED strip will fade back and forth from orange to white for at least 500ms. Notice the "&" before "FadeActive". That´s important!!!

4.3 *Class Std12VOutput*

This class provides functions to activate standard 12V output devices like lamps, LEDs, motors, shaker motors etc.

4.3.1 Instantiation

For every device you want to control from the PinLightShield you have to first call its instantiator. That call has to happen before the setup() function in your sketch.

On creation this class takes just one parameter, the pin on which it connects to the PinLightShield. The values for "pin" can be 3, 5, 6, 9, 10 or 11 if you work with the PinLightShield. Be careful when using inductive devices like motors and shaker motors. Only the pins 9, 10 and 11 are protected against such loads.

1. int pin: the PinLightShield pin that connects to the output device (3, 5, 6, 9, 10 or 11)

Example:

```
Std12VOuput MyShaker(9);        // this creates a device called MyShaker on pin 9
.                               // of the PinLightShield
.
.
.
void setup() {
}
.
.
```

4.3.2 Function Output

This functions activates the device and takes just one parameter, which specifies the intensity of the signal that will be emitted over the PWM channel. The function will constrain the value to the valid range of 0 to 255.

1. int val: the intensity of the signal (0-255)

Example:

```
MyShaker.Output(100);           // will output a medium strength signal to the device

MyShaker.Output(300);           // will output the max value of 255 to the device
```

4.3.3 Function OutputWithDelay

This function works similar to "Output", but will maintain that signal for a specified duration. Therefore this function needs 3 parameters.

1. int val: the intensity of the signal (0-255)
2. int delay: the duration of the signal in ms
3. unsigned long CurrentMillis: the current time on ms (must be determined before the call to this function by calling the Arduino function millis().
4. Boolean OutPutActive: this parameter is used as an input and is written as an output, therefore it must be called with an & in front. This variable is used to make sure that the output stays active for a minimum period of time.

Example:

```
unsigned long CurrentMillis;    // global variable stores the current time in ms
Boolean SignalActive;
Std12VOuput MyShaker(9);                // instantiates the shaker
.
```

```
void setup() {
.
.
}

void loop() {
CurrentMillis = millis();                // store the current time in ms
.
.
if (MySignal.ReadInput() || SignalActive)  // if signal active and mimimum duration not over
  {
  MyShaker.OutputWithDelay(50, 200, CurrentMillis, &SignalActive);
  }
.
}
```

This program will activate the Shaker whenever "MySignal" is active, but will activate the Shaker for at least 200ms.

4.3.4 Function MakeFlashes

This function works the same as the function in class RGBStrip (see 4.2.5).

4.4 Class Switch

This class provides functions needed to interact with mechanical playfield switches. For opto switches there´s a different class (see 4.5).

4.4.1 Instantiation

For every switch you want to scan with the PinLightShield you have to first call its instantiator. That call has to happen before the setup() function in your sketch.

On creation this class takes 2 parameters, the second one being *optional*.

1. int pin: the PinLightShield pin that connects to the switch
2. *int switchwait*: the PinLightShield will wait a certain time (in ms) before it actually registers that switch as closed[4]

Example:

[4] e.g. useful for switches in the ball trough where a ball rolls over some switches before it stops in its correct position. You´re probably only interested in that last switch where the ball finally ends up.

```
Switch MySwitch(A1);    // this creates a switch called MySwitch on pin A1 (switchwait defaults to 0)
Switch YourSwitch(A2, 50 );       // this creates a switch called YourSwitch on pin A2 (switchwait is 50ms)
.
.
.
void setup() {
}
```

4.4.2 Function ReadSwitch

This function returns "true" for a closed switch and "false" for an open one. The parameter "switchwait" is neglected in this function.

Example:

```
if (MySwitch.ReadSwitch() == true) {
.
.
}
```

4.4.3 Function ReadSwitchDelayed

This function takes the parameter "switchwait" into account and only reports a switch as "closed" when it is closed for at least that period of time. Therefore this function needs the current time as its only parameter when it´s called.

1. unsigned long CurrentMillis: the current time in ms (must be determined before the call to this function by calling the Arduino function millis().

Example:

```
unsigned long CurrentMillis;      // global variable stores the current time in ms
Switch MySwitch(9, 50);           // instantiates the switch with a switchwait of 50ms
.
void setup() {
.
.
}

void loop() {
CurrentMillis = millis();             // store the current time in ms
if (MySwitch.ReadSwitchDelayed(CurrentMillis) == true) {          // will be true if switch closed for at
                                                                  // least 50ms
  .
  .
  }
}
```

4.5 Class OptoSwitch

The class OptoSwitch implements exactly the same functions as the class Switch. Only its “Read…” functions act differently as they return a “true” when the switch is actually open and a “false” when the switch is closed. This is due to the fact that a ball going through an opto switch indeed opens the opto instead of closing it like on mechanical switches.

Who finds that confusing can still use the class Switch for optos.

4.6 Class Insert

This class provides functions to analyze the state of inserts and other lamps in the switch matrix.

4.6.1 Instantiation

For every lamp that you want to scan with the PinLightShield you have to first call its instantiator. That call has to happen before the setup() function in your sketch.

On creation this class takes 4 parameters, the last 2 being optional and only needed if you want to use the function GetBlinkInsertState (see below).

1. int pin: the PinLightShield pin that connects to the lamp
2. int FilterDelay: this deals with the fact, that a lamp in the lamp matrix is not really constantly on when it´s "ON". It rather gets quick pulses (e.g. every ~20ms on a WPC pin) which make it seem to be "ON". With this Delay we filter out those pulses so that the function InsertOn returns a constant "true" when the lamp is ON.
3. *int InsertOnDelay*: time to wait before we can say that an Insert changed from "Blinking" to "On" (value depends on the frequency in which the insert flashes). If the value of this variable is chosen too low the function GetBlinkInsertState may detect an ON-state when actually the insert is still flashing.
4. *int InsertOffDelay*: time to wait before we can say that an Insert changed from "Blinking" to "Off" (value depends on the frequency in which the insert flashes). If the value of this variable is chosen too low the function GetBlinkInsertState may detect an OFF-state when actually the insert is still flashing.

Example:

```
Insert MyInsert(4, 25);   // this creates a lamp MyInsert on pin D4 without the optional parameters
Insert YourInsert(7, 25, 200, 200);           // this creates a lamp YourInsert on pin D7 and sets it up to
                                              // detect a flashing signal of <200ms period length
```

4.6.2 Function InsertOn()

This function reads the value of the lamp and returns a “true” when the lamp is “ON” and a “false” if the lamp is “OFF”. The function takes the current time in ms as its only parameter.

1. unsigned long CurrentMillis: the current time on ms (must be determined before the call to this function by calling the Arduino function millis().

Example:

```
unsigned long CurrentMillis;      // global variable stores the current time in ms
Insert MyInsert(7, 25);           // instantiates the lamp on a WPC or WPC95 machine
.
void setup() {
.
.
}

void loop() {
CurrentMillis = millis();         // store the current time in ms
if (MyInsert.InsertOn(CurrentMillis) == true) {              // will be true if the insert is ON
  .
  .
  }
}
```

4.6.3 Function GetBlinkInsertState

This function detects the state of a lamp, that can not only be ON or OFF, but also blinking. It returns the result in a byte value with the following return codes:

0:	OFF
1:	ON
2:	BLINKING
3:	UNCLEAR (not yet clear wether Insert is ON or BLINKING

The function takes the current time in ms as its only parameter.

1. unsigned long CurrentMillis: the current time on ms (must be determined before the call to this function by calling the Arduino function millis().

Example:

```
unsigned long CurrentMillis;      // global variable stores the current time in ms
Insert MyInsert(7, 25, 200, 200);              // instantiates the lamp on a WPC or WPC95 machine
.
void setup() {
```

```
.
.
}

void loop() {
CurrentMillis = millis();             // store the current time in ms
switch (MyInsert.GetBlinkInsertState(CurrentMillis) {
  case 0:        // insert is OFF
    .
    .
    break;
  case 1:        // insert is ON
    .
    .
    break;
  case 2:        // insert is BLINKING
    .
    .
    break;
  case 3:        // insert is UNCLEAR
    .
    .
    break;
  }
}
```

4.7 *Class StdInput*

This simple class provides functions to read the state of standard input devices like flashers, coils, motors and shakers.

4.7.1 Instantiation

The class only takes one parameter on its creation.

1. int pin: the PinLightShield pin that connects to the lamp

Example:

```
StdInput MyInput(4);    // this creates a standard input device MyInput on pin D4
```

4.7.2 Function ReadInput

This function returns “true” if the device is activated and “false” if not.

Example:

```
Insert MyInput(4);                        // instantiates the input device on pin D4
.
void setup() {
.
.
}

void loop() {
if (MyInput.ReadInput() == true) {                                         // will be true if the device is ON
  .
  .
  }
}
```