ArduCor

v2.9.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 Getters and Setters

Functions

- bool ArduCor::setMainColor (uint8_t r, uint8_t g, uint8_t b)
- void ArduCor::setColor (uint16_t colorIndex, uint8_t r, uint8_t g, uint8_t b)
- void ArduCor::setCustomColorCount (uint8_t count)
- boolean ArduCor::isOn ()
- uint8_t ArduCor::customColorCount ()
- void ArduCor::brightness (uint8_t brightness)
- int ArduCor::brightness ()
- Color ArduCor::mainColor ()
- Color ArduCor::color (uint16_t i)
- uint8_t ArduCor::red (uint16_t i)
- uint8_t ArduCor::green (uint16_t i)
- uint8_t ArduCor::blue (uint16_t i)

4.1.1 Detailed Description

These are the getters and setters for ArduCor that are used to control the settings and the colors.

4.1.2 Function Documentation

4.1.2.1 blue()

```
uint8_t ArduCor::blue (
            uint16_t i )
```

Retrieve the b value at a given index in the buffer.

4.1.2.2 brightness() [1/2]

```
void ArduCor::brightness (
            uint8_t brightness )
```

Set the brightness between 0 and 100. 0 is off, 100 is full brightness.

4.1.2.3 brightness() [2/2]

```
int ArduCor::brightness ( )  [inline]
```

Retrieve the brightness level, which is a value between 0 and 100 where 100 is full brightness.

4.1.2.4 color()

```
ArduCor::Color ArduCor::color (
            uint16_t i )
```

Retrieve the color at the given index.

4.1.2.5 customColorCount()

```
uint8_t ArduCor::customColorCount ( )
```

Retrieve the amount of colors that are used from the custom array.

4.1.2.6 green()

```
uint8_t ArduCor::green (
            uint16_t i )
```

Retrieve the g value at a given index in the buffer.

4.1.2.7 isOn()

```
boolean ArduCor::isOn ( )  [inline]
```

Returns true if the LEDs are on, false if they are off.

4.1.2.8 mainColor()

```
ArduCor::Color ArduCor::mainColor ( )
```

Retrieve the main color, which is used for single color routines.

4.1.2.9 red()

```
uint8_t ArduCor::red (
            uint16_t i )
```

Retrieve the r value at a given index in the buffer.

4.1.2.10 setColor()

```
void ArduCor::setColor (
            uint16_t colorIndex,
            uint8_t r,
            uint8_t g,
            uint8_t b )
```

Set the color in the custom color array at the provided index. colorIndex must be less than the size of the custom color array or else it won't have any effect.

4.1.2.11 setCustomColorCount()

```
void ArduCor::setCustomColorCount (
            uint8_t count )
```

Sets the amount of colors used in custom multi color routines. The value given must be less than the size of the custom array or else it will be set to use the entire array.

4.1.2.12 setMainColor()

```
bool ArduCor::setMainColor (
            uint8_t r,
            uint8_t g,
            uint8_t b )
```

Sets the color used for single color routines. This is automatically called by each routine. Returns false if the new main color matches the previous main color.

Returns

true if a new color is set, false if the input matches the current color.

4.2 Single Color Routines

Functions

- void ArduCor::singleSolid (uint8_t red, uint8_t green, uint8_t blue)
- void ArduCor::singleBlink (uint8_t red, uint8_t green, uint8_t blue)
- void ArduCor::singleWave (uint8_t red, uint8_t green, uint8_t blue)
- void ArduCor::singleGlimmer (uint8_t red, uint8_t green, uint8_t blue, uint8_t percent=20)
- void ArduCor::singleFade (uint8_t red, uint8_t green, uint8_t blue, bool isSine)
- void ArduCor::singleSawtoothFade (uint8_t red, uint8_t green, uint8_t blue, bool fadeIn)

4.2.1 Detailed Description

These routines each take a R, G, and B value as parameters to generate a color. This color is the only color used by the routine.

All routines except singleSolid should be called repeatedly on a loop for their full effect. The speed of the loop determines how fast the LEDs update.

4.2.2 Function Documentation

4.2.2.1 singleBlink()

```
void ArduCor::singleBlink (
            uint8_t red,
            uint8_t green,
            uint8_t blue )
```

Switches between ON and OFF states using the provided color.

Parameters

red	strength of red LED, between 0 and 255
green	strength of green LED, between 0 and 255
blue	strength of blue LED, between 0 and 255

4.2.2.2 singleFade()

```
void ArduCor::singleFade (
            uint8_t red,
            uint8_t green,
            uint8_t blue,
            bool isSine )
```

Fades the LEDs in and out based on the provided color. Can fade in two ways: linear and sine. If `isSine` is set to false, the interval between each update is constant. If `isSine` is true, a sine wave is used to generate the intervals, resulting in lights that stay on near their full brightness for longer.

Parameters

red	strength of red LED, between 0 and 255
green	strength of green LED, between 0 and 255
blue	strength of blue LED, between 0 and 255
isSine	if true, a sine wave is used, if false, constant intervals are used.

4.2.2.3 singleGlimmer()

```
void ArduCor::singleGlimmer (
             uint8_t red,
             uint8_t green,
             uint8_t blue,
             uint8_t percent = 20 )
```

Set every LED to the provided color. A subset of the LEDs based on the percent parameter will be less bright than the rest of the LEDs.

Parameters

red	strength of red LED, between 0 and 255
green	strength of green LED, between 0 and 255
blue	strength of blue LED, between 0 and 255
percent	determines how many LEDs will be slightly dimmer than the rest, between 0 and 100

4.2.2.4 singleSawtoothFade()

```
void ArduCor::singleSawtoothFade (
             uint8_t red,
             uint8_t green,
             uint8_t blue,
             bool fadeIn )
```

If `fadeIn` is true, the LEDs start with a brightness value of 0 and each update rasies the brightness by a constant value. When it reaches maximum brightness, it resets the brightness back to 0 and repeats the fade in. If `fadeIn` is set to false, it does the opposite; it starts at full brightness and then fades to darkness.

Parameters

red	strength of red LED, between 0 and 255
green	strength of green LED, between 0 and 255
blue	strength of blue LED, between 0 and 255
fade↩ In	if true, it fades from darkness to maximum brightness, if false, it fades from maximum brightness to darkness.

4.2.2.5 singleSolid()

```
void ArduCor::singleSolid (
            uint8_t red,
            uint8_t green,
            uint8_t blue )
```

Set every LED to the provided color.

Parameters

red	strength of red LED, between 0 and 255
green	strength of green LED, between 0 and 255
blue	strength of blue LED, between 0 and 255

4.2.2.6 singleWave()

```
void ArduCor::singleWave (
            uint8_t red,
            uint8_t green,
            uint8_t blue )
```

Uses the provided color and generates groups of the color in increasing levels of brightness. On each update, the LEDs move one index to the right. This creates a wave/scrolling effect.

Parameters

red	strength of red LED, between 0 and 255
green	strength of green LED, between 0 and 255
blue	strength of blue LED, between 0 and 255

4.3 Multi Colors Routines

Functions

- void ArduCor::multiGlimmer (EPalette palette, uint8_t percent=20)
- void ArduCor::multiFade (EPalette palette)
- void ArduCor::multiRandomIndividual (EPalette palette)
- void ArduCor::multiRandomSolid (EPalette palette)
- void ArduCor::multiBars (EPalette palette, uint8_t barSizeSetting)

4.3.1 Detailed Description

These routines use multiple colors. They all take the parameter of `palette` which is used to determine which set of colors to use. The custom color array is eCustom, all other values for `palette` come from groups of preset colors. Go to the project's github for a full list of the palettes and their corresponding values.

All routines except multiBarsSolid should be called repeatedly on a loop for their full effect. The speed of the loop determines how fast the LEDs update.

4.3.2 Function Documentation

4.3.2.1 multiBars()

```
void ArduCor::multiBars (
            EPalette palette,
            uint8_t barSizeSetting )
```

Uses the chosen palette to set the LEDs in alternating patches with a size of barSize. On each update, the bars move up one LED index on each frame update to create a "scrolling" effect.

Parameters

palette	the palette to use for the routine. eCustom is the custom array, all other values are preset groups.
barSize	how many LEDs before switching to the other bar.

4.3.2.2 multiFade()

```
void ArduCor::multiFade (
            EPalette palette )
```

Fades between all the colors used by the palette.

Parameters

palette	the palette to use for the routine. eCustom is the custom array, all other values are preset groups.

4.3.2.3 multiGlimmer()

```
void ArduCor::multiGlimmer (
            EPalette palette,
            uint8_t percent = 20 )
```

This method uses its percent parameter to dim LEDs randomly, similar to the standard glimmer mode. It also uses the percent to randomly change the color of select LEDs to a color in the chosen palette. The base color is the first from the chosen palette.

Parameters

palette	the palette to use for the routine. eCustom is the custom array, all other values are preset groups.
percent	percent of LEDs that will get the glimmer applied, between 0 and 100

4.3.2.4 multiRandomIndividual()

```
void ArduCor::multiRandomIndividual (
            EPalette palette )
```

sets each individual LED as a random color from the chosen palette.

Parameters

palette	the color array to use for the routine. eCustom is the custom array, all other values are palette arrays.

4.3.2.5 multiRandomSolid()

```
void ArduCor::multiRandomSolid (
            EPalette palette )
```

A random color is chosen from the chosen palette and applied to each LED.

Parameters

palette	the palette to use for the routine. eCustom is the custom array, all other values are preset groups.

4.4 Post Processing

These methods can be called after a routine is chosen but before the routines get displayed to the LEDs. They add special effects to the routines.

Chapter 5

Class Documentation

5.1 ArduCor Class Reference

An Arduino library that provides a set of RGB lighting routines for compatible LED array hardware.

`#include <ArduCor.h>`

Public Member Functions

- ArduCor (uint16_t ledCount)
- void resetToDefaults ()
- void turnOn ()
- void turnOff ()
- bool setMainColor (uint8_t r, uint8_t g, uint8_t b)
- void setColor (uint16_t colorIndex, uint8_t r, uint8_t g, uint8_t b)
- void setCustomColorCount (uint8_t count)
- boolean isOn ()
- uint8_t customColorCount ()
- void brightness (uint8_t brightness)
- int brightness ()
- Color mainColor ()
- Color color (uint16_t i)
- uint8_t red (uint16_t i)
- uint8_t green (uint16_t i)
- uint8_t blue (uint16_t i)
- void singleSolid (uint8_t red, uint8_t green, uint8_t blue)
- void singleBlink (uint8_t red, uint8_t green, uint8_t blue)
- void singleWave (uint8_t red, uint8_t green, uint8_t blue)
- void singleGlimmer (uint8_t red, uint8_t green, uint8_t blue, uint8_t percent=20)
- void singleFade (uint8_t red, uint8_t green, uint8_t blue, bool isSine)
- void singleSawtoothFade (uint8_t red, uint8_t green, uint8_t blue, bool fadeIn)
- void multiGlimmer (EPalette palette, uint8_t percent=20)
- void multiFade (EPalette palette)
- void multiRandomIndividual (EPalette palette)
- void multiRandomSolid (EPalette palette)
- void multiBars (EPalette palette, uint8_t barSizeSetting)
- void applyBrightness ()
- bool drawColor (uint16_t i, uint8_t red, uint8_t green, uint8_t blue)

5.1.1 Detailed Description

An Arduino library that provides a set of RGB lighting routines for compatible LED array hardware.

Version

v3.0.0

Date

April 14, 2018

Author

Tim Seemann

Copyright

MIT License

This library has been tested with SeeedStudio Rainbowduinos, quite a few of the Adafruit Neopixels products, and a standard RGB LED. Sample code is provided in the git repo for all tested hardware in the samples folder of the git repository.

If you are starting a project from scratch, first you'll need to make a global object in the arduino sketch:

```
ArduCor routines = ArduCor(LED_COUNT);
```

where `LED_COUNT` is the number of LEDs in your array.

The library produces lighting routines based on the functions used and stores the routine in its internal buffers. These buffers can then be accessed by getters and displayed on the LED hardware. For routines that change over time, this process should be repeated on a loop. For example, here is how you would make a red blinking light with the library and a Neopixels board:

First, call this function to store the routine in the library's internal buffers:

```
routines.singleBlink(255, 0, 0);
```

Then, update the LED array with the values from the library's RGB buffer. The way to do this will vary from hardware to hardware, but for a NeoPixels sample, it would look something like this:

```
routines.applyBrightness();  // Optional. Dims the LEDs based on the routines.brightness()
       setting
for (int x = 0; x < LED_COUNT; x++) {
   pixels.setPixelColor(x, pixels.Color(routines.red(x),
                                        routines.green(x),
                                        routines.blue(x)));
}
pixels.show();
```

By this point, the LEDs should be showing red. To achieve the blink effect, put both of these in your `loop()` function and then put a delay between updates. This delay will be used to determine how fast the LED's blink.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 ArduCor()

```
ArduCor::ArduCor (
             uint16_t ledCount )
```

Required constructor. The library should be stored in global memory and allocated only once at startup.

It will allocate `4 * ledCount` bytes.

Parameters

ledCount	number of individual RGB LEDs.

5.1.3 Member Function Documentation

5.1.3.1 applyBrightness()

```
void ArduCor::applyBrightness ( )
```

This function takes the brightness() value given to the routines object and applies it to every LED. Relatively speaking, this is a pretty expensive operation so it is left optional.

5.1.3.2 drawColor()

```
bool ArduCor::drawColor (
            uint16_t i,
            uint8_t red,
            uint8_t green,
            uint8_t blue )
```

Attempts to draw the color provided on the index provided.

Parameters

i	the index of the LED that you want to change. Must be less than the total amount of LEDs or else it will return false.
red	the new red value of the LED, between 0 and 255.
green	the new green value of the LED, between 0 and 255.
blue	the new blue value of the LED, between 0 and 255.

Returns

true if index exists and the color was drawn, false otherwise.

5.1.3.3 resetToDefaults()

```
void ArduCor::resetToDefaults ( )
```

Resets all internal values to the original values.

5.1.3.4 turnOff()

```
void ArduCor::turnOff ( )
```

Turns off all the LEDs. To turn the lights back on, call any light routine or call `turnOn()`.

5.1.3.5 turnOn()

```
void ArduCor::turnOn ( )
```

Turns on all the LEDs.

Chapter 6

File Documentation

6.1 ArduCorProtocols.h File Reference

Enumerations

- enum ERoutine {
  eSingleSolid, eSingleBlink, eSingleWave, eSingleGlimmer,
  eSingleFade, eSingleSawtoothFade, eMultiGlimmer, eMultiFade,
  eMultiRandomSolid, eMultiRandomIndividual, eMultiBars }
- enum EPalette {
  eCustom, eWater, eFrozen, eSnow,
  eCool, eWarm, eFire, eEvil,
  eCorrorsive, ePoison, eRose, ePinkGreen,
  eRedWhiteBlue, eRGB, eCMY, eSixColor,
  eSevenColor }
- enum EPacketHeader {
  eOnOffChange, eModeChange, eCustomArrayColorChange, eBrightnessChange,
  eCustomColorCountChange, eIdleTimeoutChange, eStateUpdateRequest, eCustomArrayUpdateRequest }

6.1.1 Detailed Description

Version

v3.0.0

Date

April 14, 2018

Author

Tim Seemann

Copyright

This file defines the protocols used for the sample sketches.

This file also gets copied to other projects as part of integrating with this project. For example, the Corluma project has a C++ version of this file. If packets between the two projects seem mixed up, check that the version of the Corluma App you are using matches the version of the your ArduCor library.

Protocol Version: 3.3

6.1.2 Enumeration Type Documentation

6.1.2.1 EPacketHeader

`enum EPacketHeader`

Message headers for packets coming over serial.

Enumerator

eOnOffChange	**0** *Takes one parameter, 0 turns off, 1 turns on.*
eModeChange	**1** *Takes multiple parameters depending on the use case. Changes the lighting routine currently getting displayed.*
eCustomArrayColorChange	**2** *Takes four parameters. The first is the index of the custom color, the remaining three parameters are a 0-255 representation of Red, Green, and Blue.*
eBrightnessChange	**3** *Takes one parameter, sets the brightness between 0 and 100.*
eCustomColorCountChange	**4** *Change the number of colors used in a custom array routine.*
eIdleTimeoutChange	**5** *Set to 0 to turn off, set to any other number minutes until idle timeout happens.*
eStateUpdateRequest	**6** *Sends back a packet that contains basic LED state information.*
eCustomArrayUpdateRequest	**7** *Sends back a packet that contains the size of the custom array and all of the colors in it.*

6.1.2.2 EPalette

`enum EPalette`

used during multi color routines to determine which colors to use in the routine. eCustom uses the custom color array, eAll generates its colors randomly. All other values use presets based around overall themes.

Enumerator

eCustom	**0** *Use the custom color array instead of a preset group.*
eWater	**1** *Shades of blue with some teal.*
eFrozen	**2** *Shades of teal with some blue, white, and light purple.*

Enumerator

eSnow	**3** *Shades of white with some blue and teal.*
eCool	**4** *Based on the cool colors: blue, green, and purple.*
eWarm	**5** *Based on the warm colors: red, orange, and yellow.*
eFire	**6** *Similar to the warm set, but with an emphasis on oranges to give it a fire-like glow.*
eEvil	**7** *Mostly red, with some other, evil highlights.*
eCorrorsive	**8** *Greens and whites, similar to radioactive goo from a 90s kids cartoon.*
ePoison	**9** *A purple-based theme. Similar to poison vials from a 90s kids cartoon.*
eRose	**10** *Shades of pink, red, and white.*
ePinkGreen	**11** *The colors of watermelon candy. bright pinks and bright green.*
eRedWhiteBlue	**12** *Bruce Springsteen's favorite color scheme, good ol' red, white, and blue.*
eRGB	**13** *red, green, and blue.*
eCMY	**14** *Cyan, magenta, yellow.*
eSixColor	**15** *Red, yellow, green, cyan, blue, magenta.*
eSevenColor	**16** *Red, yellow, green, cyan, blue, magenta, white.*

6.1.2.3 ERoutine

`enum ERoutine`

Each routine makes the LEDs shine in different ways. There are two main types of routines: Single Color Routines use a single color while Multi Color Routines rely on an EPalette.

Enumerator

eSingleSolid	**0** *Shows a single color at a fixed brightness.*
eSingleBlink	**1** *Alternates between showing a single color at a fixed brightness and turning the LEDs completely off.*
eSingleWave	**2** *Linear fade of the brightness of the LEDs.*
eSingleGlimmer	**3** *Randomly dims some of the LEDs to give a glimmer effect.*

Enumerator

eSingleFade	**4** *Fades the brightness in and out of the LEDs. Takes a parameter of either 0 or 1. If its 0, it fades linearly. If its 1, it fades using a sine wave, so less time in the mid range of brightness and more time in the full and very dim light.*
eSingleSawtoothFade	**5** *fades in or out using a sawtooth function. Takes a parameter of either 0 or* 1. *If its 0, it starts off and fades to full brightness. If its 1, it starts at full brightness and fades to zero. The fade is linear.*
eMultiGlimmer	**6** *Uses the first color of the array as the base color and uses the other colors for a glimmer effect.*
eMultiFade	**7** *Fades slowly between each color in the array.*
eMultiRandomSolid	**8** *Chooses a random color from the array and lights all all LEDs to match that color.*
eMultiRandomIndividual	**9** *Chooses a random color from the array for each individual LED.*
eMultiBars	**10** *Draws the colors of the array in alternating groups of equal size. On each update, it moves those groups one index to the right, creating a scrolling effect.*

6.2 Palettes.h File Reference

```
#include <avr/pgmspace.h>
```

6.2.1 Detailed Description

Version

v3.0.0

Date

April 14, 2018

Author

Tim Seemann

Copyright

MIT License

These color palettes are stored in program memory and loaded into a buffer when accessed. This makes the presets read-only, but in return, it allows them to take a much smaller hit on SRAM usage.

Index