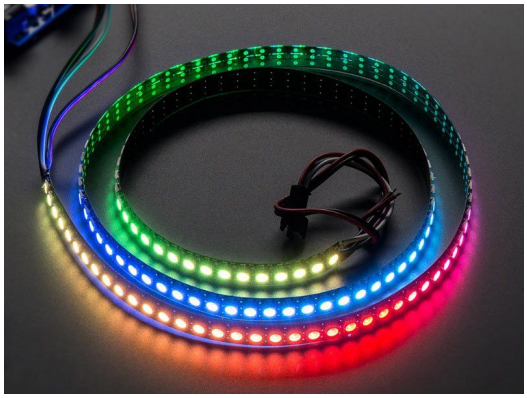
Adafruit NeoPixel : Arduino Library Use



It's assumed at this point that you have the Adafruit_NeoPixel library for Arduino installed and have run the strandtest example sketch successfully.

To learn about writing your own NeoPixel sketches, let's begin by **dissecting the strandtest sketch**…

All NeoPixel sketches begin by including the header file:

```
#include <Adafruit_NeoPixel.h>
```

The block of code that follows is mostly descriptive comments. Only the last line is really doing any work:

```
#define PIN 6


// Parameter 1 = number of pixels in strip
// Parameter 2 = pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, PIN, NEO_GRB + NEO_KHZ800);
```

The first line assigns a number to the symbol "PIN" for later reference. It doesn't *need* to be done this way, but makes it easier to change the pin where the NeoPixels are connected without digging deeper into the code.

The last line declares a NeoPixel *object.* We'll refer to this by name later to control the strip of pixels. There are three parameters or *arguments* in parenthesis:

1. The number of sequential NeoPixels in the strip. In the example this is set to 60, equal to 1 meter of medium-density strip. Change this to match the actual number you're using.
2. The pin number to which the NeoPixel strip (or other device) is connected. Normally this would be a number, but we previously declared the symbol PIN to refer to it by name here.
3. A value indicating the type of NeoPixels that are connected. **In most cases you can leave this off and pass just two arguments;** the example code is just being extra descriptive. If you have a supply of classic "V1" Flora pixels, those require NEO_KHZ400 + NEO_RGB to be passed here.

For through-hole 8mm NeoPixels, use NEO_RGB instead of NEO_GRB in the strip declaration. For RGBW LEDs use NEO_RGBW (some RGBW strips use NEO_GRBW, so try that if you're getting unexpected results!)

Then, in the setup() function, call begin() to prepare the data pin for NeoPixel output:

```
void setup() {
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}
```

The second line, strip.show(), isn't absolutely necessary, it's just there to be thorough. That function pushes data out to the pixels…since no colors have been set yet, this initializes all the NeoPixels to an initial "off" state in case some were left lit by a prior program.
In the strandtest example, loop() doesn't set any pixel colors on its own — it calls other functions that create animated effects. So let's ignore it for now and look ahead, inside the individual functions, to see how the strip is controlled.

There are two ways to set the color of a pixel. The first is:

```
strip.setPixelColor(n, red, green, blue);
```

or, if you're using RGBW strips:

```
strip.setPixelColor(n, red, green, blue, white);
```

The first argument — **n** in this example — is the pixel number along the strip, starting from 0 closest to the Arduino. If you have a strip of 30 pixels, they're numbered 0 through 29. It's a computer thing. You'll see various places in the code using a for loop, passing the loop counter variable as the pixel number to this function, to set the values of multiple pixels.

The next three arguments are the pixel color, expressed as red, green and blue brightness levels, where 0 is dimmest (off) and 255 is maximum brightness. The last *optional* argument is for white, which will only be used if the strip was defined during creation as an RGBW type and the strip actually is RGBW type.

To set the 12th pixel (#11, counting from 0) to magenta (red + blue), you could write:

```
    strip.setPixelColor(11, 255, 0, 255);
```

to set the 8th pixel (#7 counting from 0) to half-brightness white, with no light from red/green/blue, use:

```
strip.setPixelColor(7, 0, 0, 0, 127);
```

An alternate syntax has just two arguments:

```
strip.setPixelColor(n, color);
```

Here, color is a 32-bit type that merges the red, green and blue values into a single number. This is sometimes easier or faster for some (but not all) programs to work with; you'll see the strandtest code uses both syntaxes in different places.

You can also convert separate red, green and blue values into a single 32-bit type for later use:

```
uint32_t magenta = strip.Color(255, 0, 255);
```

Then later you can just pass "magenta" as an argument to setPixelColor rather than the separate red, green and blue numbers every time.

You can also (optionally) add a white component to the color at the end, like this:

```
uint32_t greenishwhite = strip.Color(0, 64, 0, 64);
```

setPixelColor() does not have an immediate effect on the LEDs. To "push" the color data to the strip, call show():

```
strip.show();
```

This updates the whole strip at once, and despite the extra step is actually a good thing. If every call to setPixelColor() had an immediate effect, animation would appear jumpy rather than buttery smooth.
You can query the color of a previously-set pixel using getPixelColor():

```
uint32_t color = strip.getPixelColor(11);
```

This returns a 32-bit merged color value.

The number of pixels in a previously-declared strip can be queried using numPixels():

```
uint16_t n = strip.numPixels();
```

The overall brightness of all the LEDs can be adjusted using setBrightness(). This takes a single argument, a number in the range 0 (off) to 255 (max brightness). For example, to set a strip to 1/4 brightness:

```
strip.setBrightness(64);
```

Just like setPixel(), **this does not have an immediate effect.** You need to follow this with a call to show().

setBrightness() was intended to be called *once,* in setup(), to limit the current/brightness of the LEDs throughout the life of the sketch. It is *not* intended as an animation effect itself! The operation of this function is "lossy" — it modifies the current pixel data in RAM, not in the show() call — in order to meet NeoPixels' strict timing requirements. Certain animation effects are better served by leaving the brightness setting alone, modulating pixel brightness in your own sketch logic and redrawing the full strip with setPixel().

I'm calling setPixel() but nothing's happening!

There are two main culprits for this:

1. forgetting to call strip.begin() in setup().
2. forgetting to call strip.show() after setting pixel colors.

Another (less common) possibility is running out of RAM — see the last section below. If the program *sort of* works but has unpredictable results, consider that.

Can I have multiple NeoPixel objects on different pins?

Certainly! Each requires its own declaration with a unique name:

```
Adafruit_NeoPixel strip_a = Adafruit_NeoPixel(16, 5);
Adafruit_NeoPixel strip_b = Adafruit_NeoPixel(16, 6);
```

The above declares two distinct NeoPixel objects, one each on pins 5 and 6, each containing 16 pixels and using the implied default type (NEO_KHZ800 + NEO_GRB).

Can I connect multiple NeoPixel strips to the same Arduino pin?

In many cases, yes. All the strips will then show exactly the same thing. This only works up to a point though…four strips on a single pin is a good and reliable number. If you need more than that, individual NeoPixels can be used as buffers to "fan out" to more strips: connect one Arduino pin to the inputs of four separate NeoPixels, then connect each pixels' output to the inputs of four strips (or fewer, if you don't need quite that many). If the strips are 10 pixels long, declare the NeoPixel object as having 11 pixels. The extra "buffer" pixels will be at position #0 — just leave them turned off — and the strips then run from positions 1 through 10.

I'm getting the wrong colors. Red and blue are swapped!

When using through-hole 8mm NeoPixels (or V1 Flora pixels), use NEO_RGB for the third parameter in the Adafruit_NeoPixel declaration. For all other types of NeoPixels, use NEO_GRB.

The colors fall apart when I use setBrightness() repeatedly!

See note above; setBrightness() is designed as a one-time setup function, not an animation effect.

Also see the "Advanced Coding" page — there's an alternative library that includes "nondestructive" brightness adjustment, among other features!

Pixels Gobble RAM

Each NeoPixel requires about 3 bytes of RAM. This doesn't sound like very much, but when you start using dozens or even hundreds of pixels, and consider that the mainstream Arduino Uno only has 2 kilobytes of RAM (often much less after other libraries stake their claim), this can be a real problem!

For using really large numbers of LEDs, you might need to step up to a more potent board like the Arduino Mega or Due. But if you're close and need just a little extra space, you can sometimes tweak your code to be more RAM-efficient.