



Adafruit DotStar LEDs

Created by Phillip Burgess

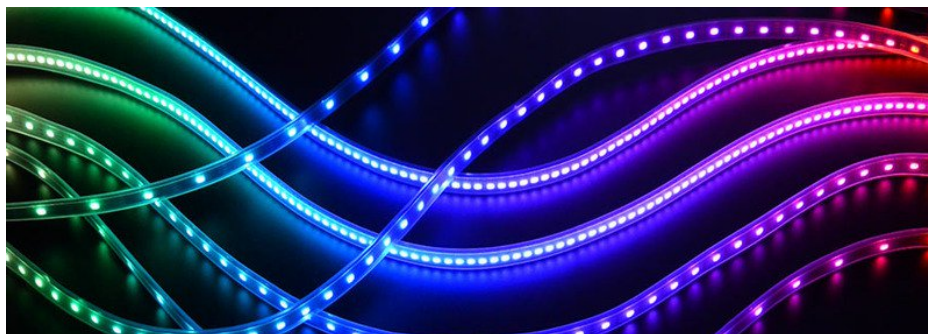


Last updated on 2018-10-09 05:36:31 PM UTC

Guide Contents

Guide Contents	2
Overview	3
DotStars vs NeoPixels	3
DotStars	3
NeoPixels	3
Form Factors	4
DotStar Strips	5
RGB DotStar Strips	5
White DotStar Strips	7
Finer Details About Flexible DotStar Strips	9
Rigid Ultra High Density DotStar LED PCB Bar	9
DotStar Matrices	11
Flexible DotStar Matrices	11
Rigid DotStar Matrices	12
Finer Details About DotStar Matrices	13
Individual DotStars	15
SMT DotStars	15
Power and Connections	17
Powering DotStar LEDs	17
Connecting DotStar LEDs	17
Software	20
Arduino Library Installation	21
Install Adafruit_DotStar via Library Manager	21
Manually Install Adafruit_DotStar Library	21
A Simple Code Example: strandtest	22
Nothing happens!	22
Something happens but the LEDs are blinking in a weird way!	22
Arduino Library Use	23
I'm calling setPixel() but nothing's happening!	25
Can I have multiple DotStar objects on different pins?	25
Can I connect multiple DotStar strips to the same Arduino pins?	25
I'm getting the wrong colors. Red and blue are swapped!	25
Most NeoPixel Code Adapts Easily to DotStars	26
Pixels Gobble RAM	26
DotStarMatrix Library	27
Layouts	27
Why not just use the rotation feature in Adafruit_GFX?	30
Tiled Matrices	30
Other Layouts	31
RAM Again	32
Gamma Correction	32

Overview



NeoPixel LEDs are the bee's knees, but in a few scenarios they come up short...connecting odd microcontrollers that can't match their strict timing, or fast-moving persistence-of-vision displays. Adafruit DotStar LEDs deliver high speed PWM and an easy-to-drive two-wire interface, bridging the gaps in the spectrum of awesome.

DotStars vs NeoPixels

The basic idea behind DotStars and NeoPixels is the same: a continuous string of individually-addressable RGB LEDs, driven by a microcontroller. The way each goes about it is a little different. DotStars *aren't necessarily a better thing in every situation*...there are tradeoffs, each has its pros and cons to consider...

DotStars	NeoPixels
<ul style="list-style-type: none">+ Extremely fast data¹ and PWM² rates, suitable for persistence-of-vision displays.+ Easier to interface to a broader range of devices; no strict signal timing requirements.+ Works with Servo library, tone() function, interrupts, etc.– Slightly more expensive.– Fewer available form factors; strips and flex matrices only.– Needs two pins for control.	<ul style="list-style-type: none">+ More affordable.+ Wide range of form-factors (pixels, rings, matrices, etc.).+ Works from a single microcontroller pin.+ RGRW (RGB+white) variants available.– Strict 800 KHz data rate; not all systems can generate this, and speed is a bottleneck on very long strands.– 400 Hz refresh/PWM rate not suitable for persistence-of-vision effects.– Not compatible with interrupts (e.g. Arduino Servo library or tone() function).

¹ Up to 8 MHz on Arduino, up to 32 MHz on Raspberry Pi.

² 19.2 KHz.

Form Factors

DotStar products are available in numerous form factors...from individual tiny pixels to huge matrices...plus strips, FeatherWings and a few surprises.

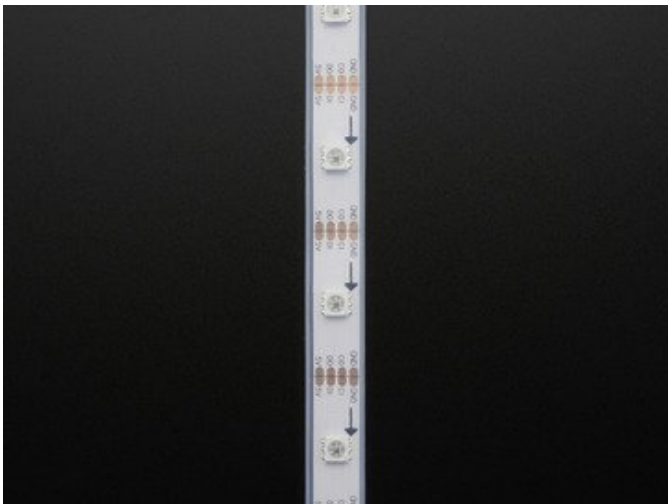
Pick a category from the left column for product links and tips & tricks specific to each type of DotStar offering.

DotStar Strips

The most popular type of DotStars are these **flexible LED strips**...they can be cut to length and fit into all manner of things. There are many varieties! Two **vital** things to be aware of:

- Though strips are described as “flexible,” **they do not tolerate continuous and repeated bending**. “Formable” might be a better word. A typical application is architecture, where they can be curved around columns and then stay put. Repeated flexing (as on costumes) will soon crack the solder connections. For wearable use, affix shorter segments to a semi-rigid base (e.g. a hat, BMX armor, etc.).
- Watch your power draw. Though each pixel only needs a little current, **it adds up fast**...DotStar strips are so simple to use, one can quickly get carried away! We’ll explain more on the “Power and Connections” page.

RGB DotStar Strips



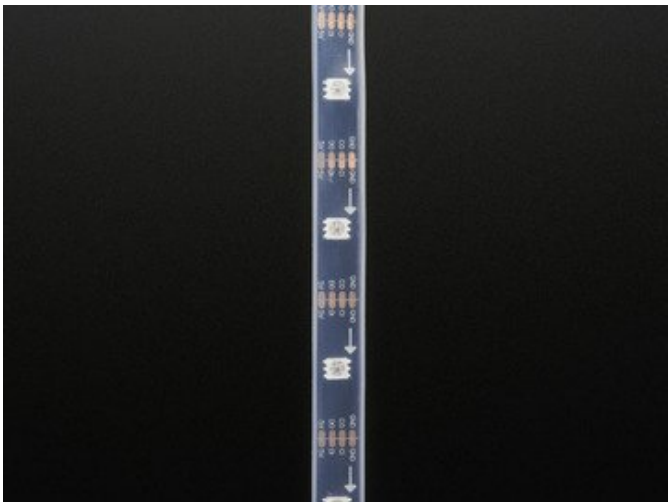
DotStar Digital RGB LED Weatherproof Strip is available in three different “densities”: 30, 60 and 144 LEDs per meter, on a white or black backing strip.

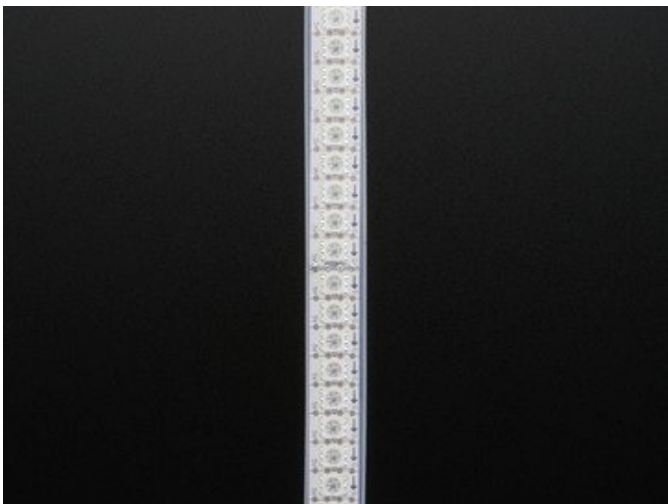
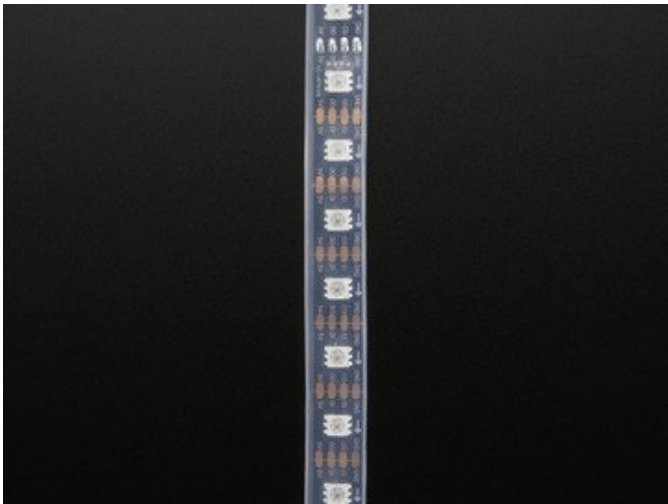
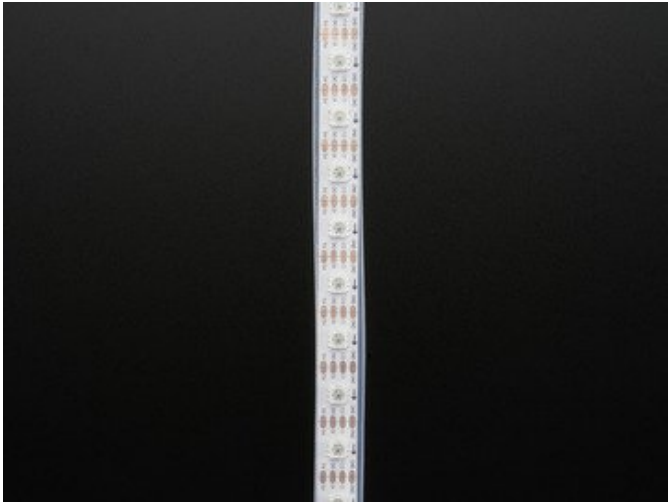
- **30 LEDs per meter, white strip** (<https://adafru.it/od1>)
- **30 LEDs per meter, black strip** (<https://adafru.it/Clk>)
- **60 LEDs per meter, white strip** (<https://adafru.it/CIl>)
- **60 LEDs per meter, black strip** (<https://adafru.it/sCQ>)
- **144 LEDs per meter, white strip** (<https://adafru.it/CIm>)
- **144 LEDs per meter, black strip** (<https://adafru.it/CIp>)
- **144/m white strip, half-meter** (<https://adafru.it/Clo>)
- **144/m black strip, half-meter** (<https://adafru.it/Clp>)

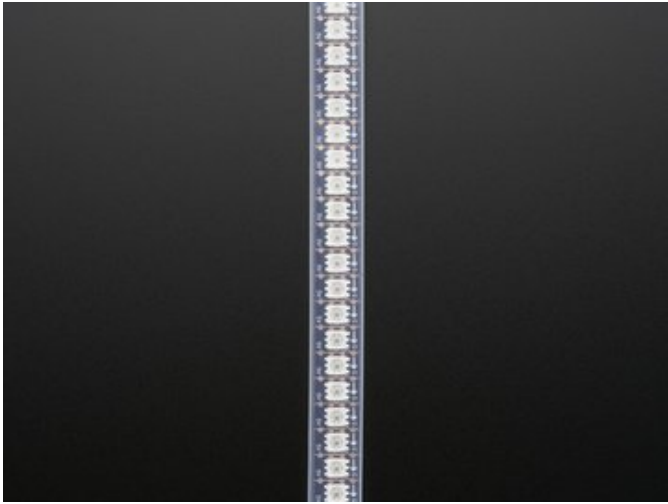
The *approximate* peak power use (all LEDs on at maximum brightness) per meter is:

- **30 LEDs:** 9 Watts (about 1.8 Amps at 5 Volts).
- **60 LEDs:** 18 Watts (about 3.6 Amps at 5 Volts).
- **144 LEDs :** 43 watts (8.6 Amps at 5 Volts).

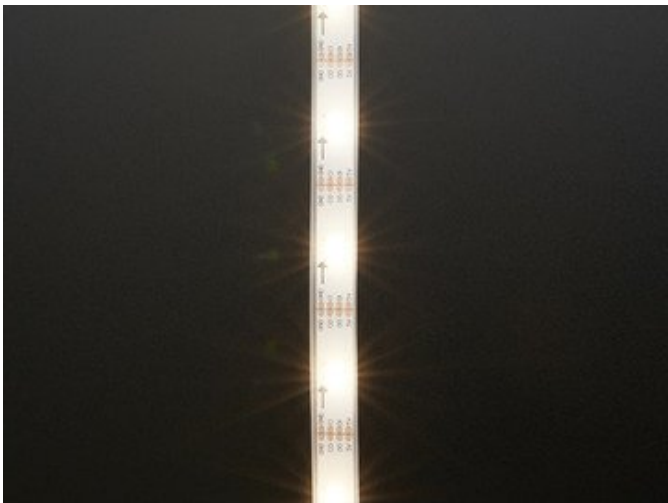
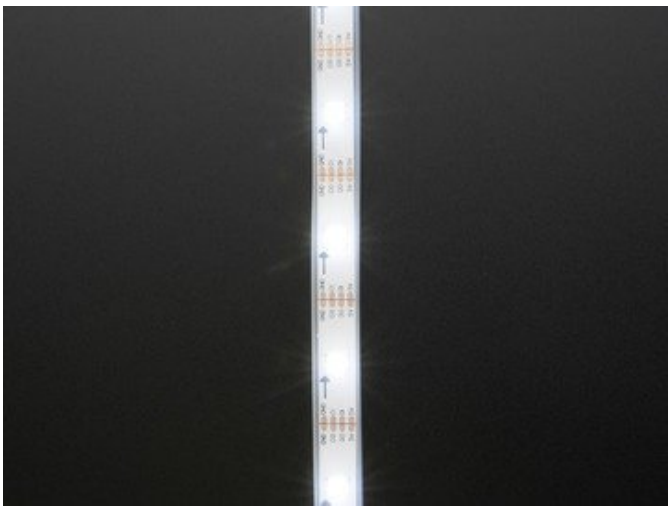
Mixed colors and lower brightness settings will use proportionally less power.







White DotStar Strips



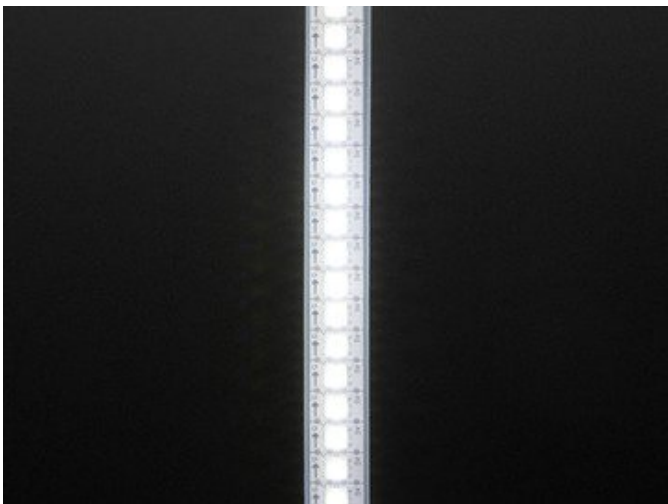
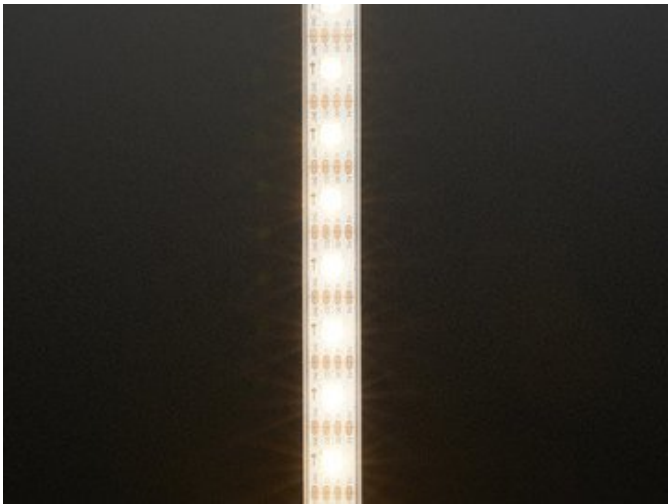
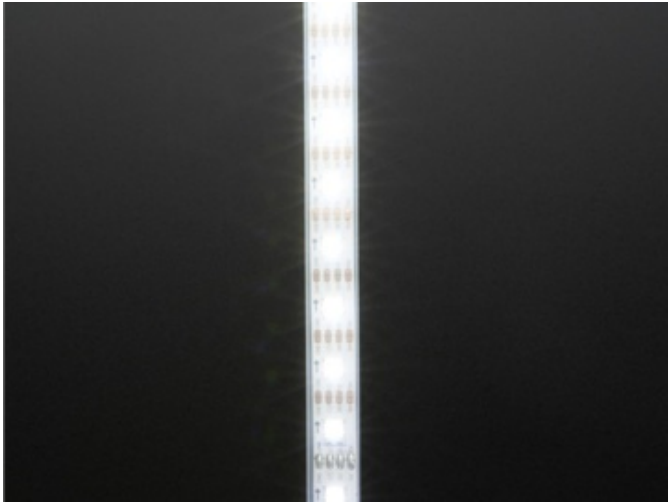
A recent addition is **White** DotStar strips. Rather than red, green and blue, these contain three identical white LED elements — either “cool” or “warm” white. For monochrome applications, white DotStars are more “true” and pleasing to the eye than white mixed from red+green+blue. Like the RGB strips, they’re available in different pixel densities.

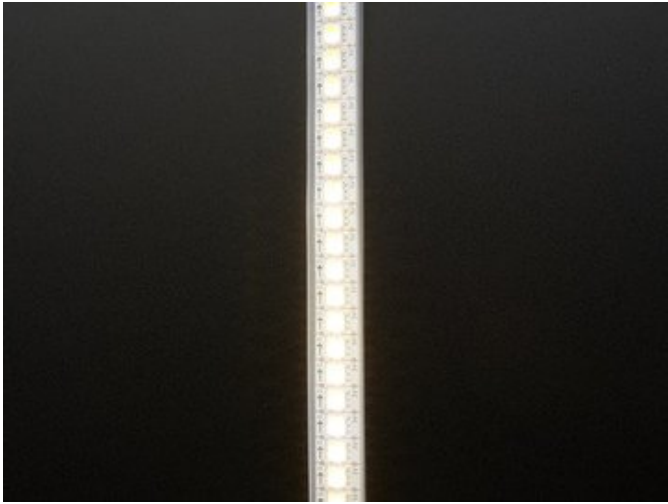
- **30 Cool White LEDs per meter** (<https://adafru.it/Clq>)
- **30 Warm White LEDs per meter** (<https://adafru.it/Clr>)
- **60 Cool White LEDs per meter** (<https://adafru.it/Cls>)
- **60 Warm White LEDs per meter** (<https://adafru.it/Clt>)
- **144 Cool White LEDs per meter** (<https://adafru.it/Clu>)
- **144 Warm White LEDs per meter** (<https://adafru.it/Clv>)

Approximate color temperature:

- Cool white: 6000–6500K
- Warm white: 3000K

White DotStars are only available with the white backing strip. Power consumption is comparable to the RGB strips. Half-meter strips not offered.

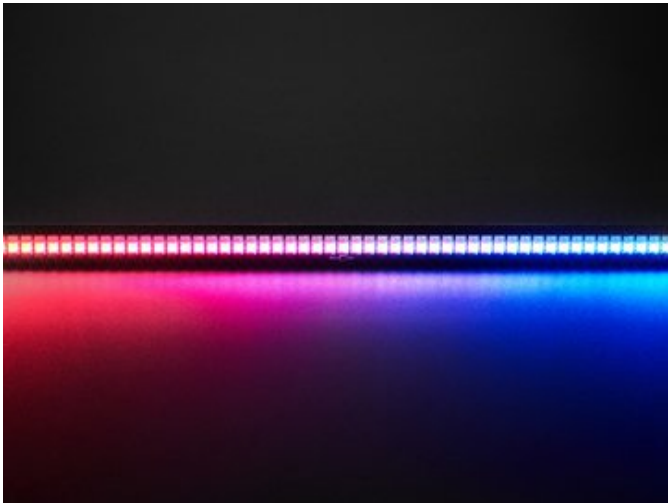




Finer Details About Flexible DotStar Strips

- **144 pixel/m** DotStar strips are sold in **one meter** (RGB or white) and **half meter** (RGB only) lengths. Each of these is a **separate** strip with end connectors. Longer contiguous lengths are *not* offered in 144 pixels/m.
- **30 and 60 pixel/m** DotStar strips are sold in **one meter multiples**. Orders for multiple meters will be a **single contiguous strip, up to a limit**: 4 meters for 60 pixels/m strip, 5 meters for 30 pixels/m.
- For **30 and 60 pixels/meter** strips, if purchasing **less than a full reel** (4 or 5 meters, respectively), the strip **may or may not** have 4-pin JST plugs soldered at one or both ends. These plugs are for factory testing and might be at **either end** — the plug does not always indicate the input end! **Arrows printed on the strip show the actual data direction. You may need to solder your own wires or plug.**
- The flex strips are **enclosed** in a weatherproof silicone sleeve, making them immune to rain and splashes, but are not recommended for continuous submersion.
- The silicone sleeve can be **cut and removed** for a slimmer profile, but this compromises the strip's weather resistance.
- **Very few glues will adhere to the weatherproof silicone sleeve.** Using zip ties for a “mechanical” bond is usually faster and easier. The only reliable glues we’ve found are Permatex 66B Clear RTV Silicone (not all silicone glues will work!) and Loctite Plastics Bonding System, a 2-part cyanoacrylate glue. Customers have reported *excellent* results with **Permatex Ultra Grey Silicone Gasket Maker** as well.
- However, **do not use Permatex 66B silicone to seal the open end of a cut strip!** Like many RTV silicones, 66B releases acetic acid when curing, which can destroy electronics. It’s fine on the *outside* of the strip, but not the *inside*. Use **GE Silicone II** for sealing strip ends, or good ol’ **hot melt glue**.
- **2-sided carpet tape** provides a light grip on the silicone sleeve; something like a Post-It Note. Or you can try **clear duct tape** over the top.
- **All LED strips** are manufactured in **1/2 meter** segments that are then **joined** into a longer strip. The pixel spacing across these joins is usually 2-3 millimeters different than the rest. This is not a manufacturing mistake, just physical reality.

Rigid Ultra High Density DotStar LED PCB Bar



Another linear form factor similar to a strip, but this one is **not flexible**. It features **128 tiny** DotStar LEDs packed into a 400 millimeter bar with wires at each end with 4-pin JST connectors.

The aluminum-backed PCB is rigid but requires support so it doesn't droop and crack. It's not intended to be bent or curved at all. Also unlike strips, this bar is **not weatherproof**.

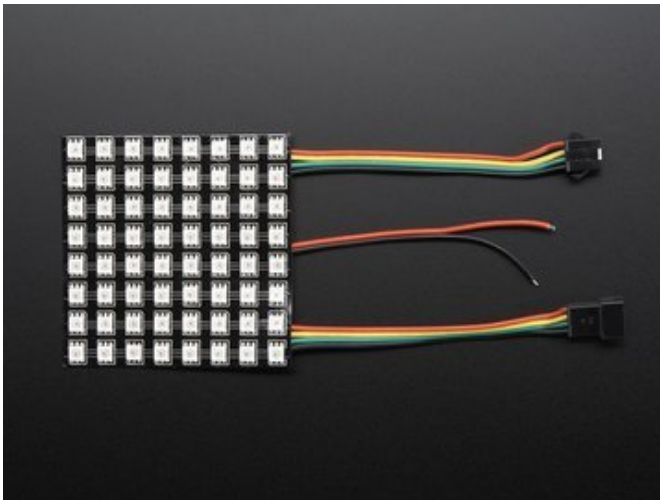
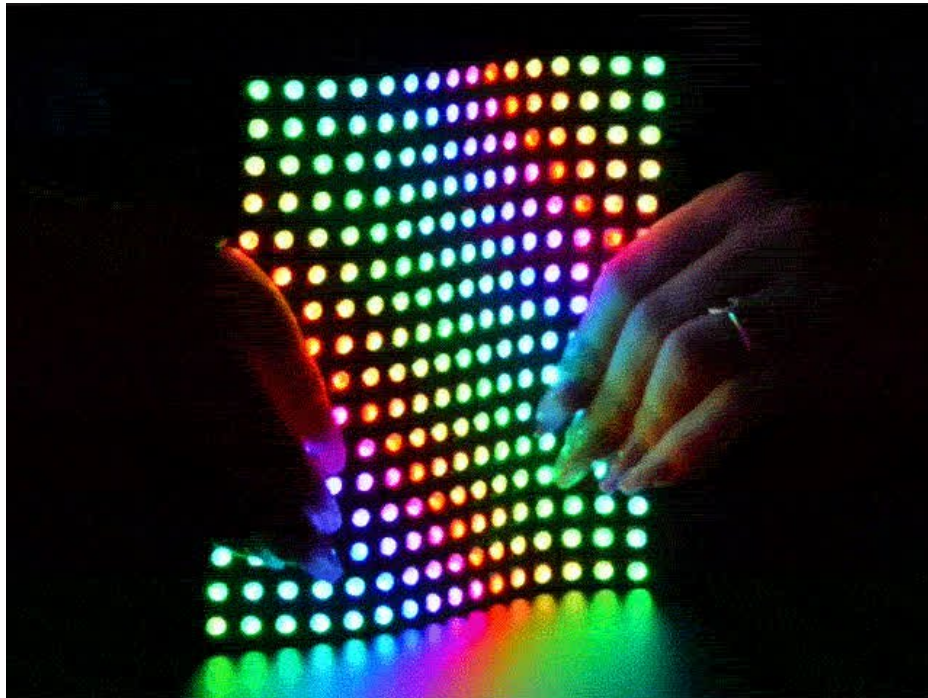
- [Ultra High Density DotStar LED PCB Bar - 128 LEDs \(https://adafru.it/Clw\)](https://adafru.it/Clw)



DotStar Matrices

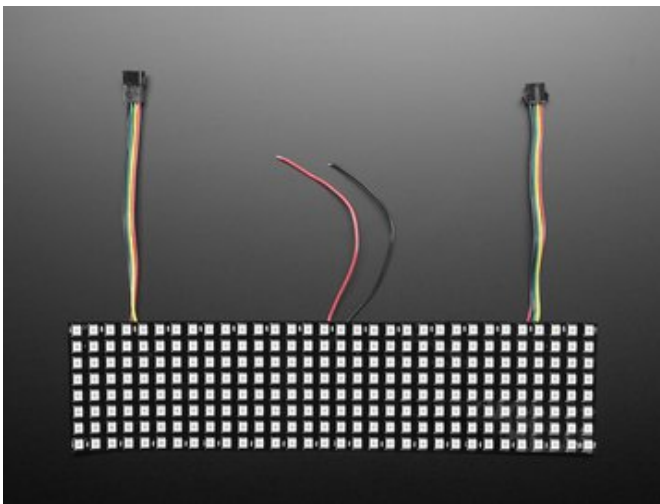
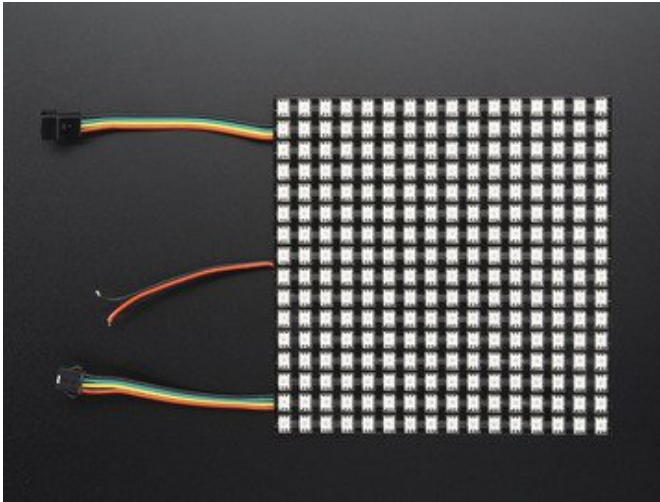
DotStar matrices are two-dimensional *grids* of DotStar LEDs, all controlled from two microcontroller pins.

Flexible DotStar Matrices



Flexible DotStar matrices are available in three different sizes:

- **8x8 RGB pixels** (<https://adafru.it/Cly>)
- **16x16 RGB pixels** (<https://adafru.it/Clz>)
- **8x32 RGB pixels** (<https://adafru.it/CIA>)



Size	Dimensions	Total # of LEDs	Max Power Draw (approx)
8x8	80 mm / 3.15" square	64	19 Watts (3.8 Amps at 5 Volts)
16x16	160 mm / 6.3" square	256	77 Watts (15 Amps at 5 Volts)
8x32	320 mm x 80 mm / 12.6" x 3"	256	77 Watts (15 Amps at 5 Volts)

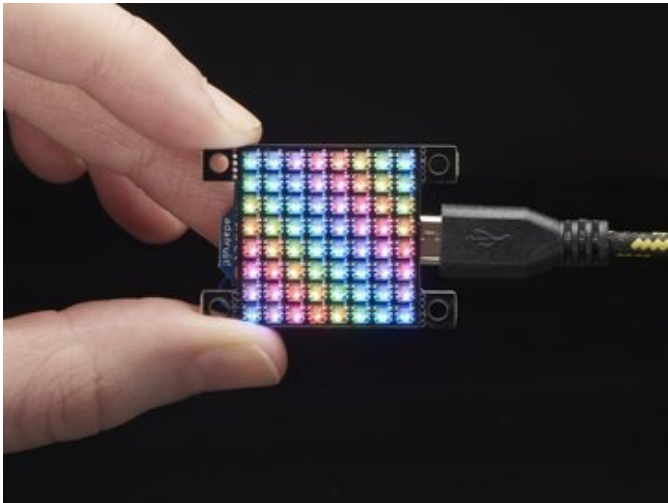
Flex matrices are about 2 millimeters (0.08 inches) thick.

Though called “flexible,” **these matrices do not tolerate continuous and repeated bending**. “Formable” might be a better word — they can be bent around a rigid or semi-rigid shape, like a hat. Repeated flexing (as on costumes) will soon crack the solder connections. (The videos on the product pages are to highlight just how flexible these matrices are, but this really is a “don’t try this at home” thing.)

Flex matrices are available with **RGB pixels only**; pure white is not offered.

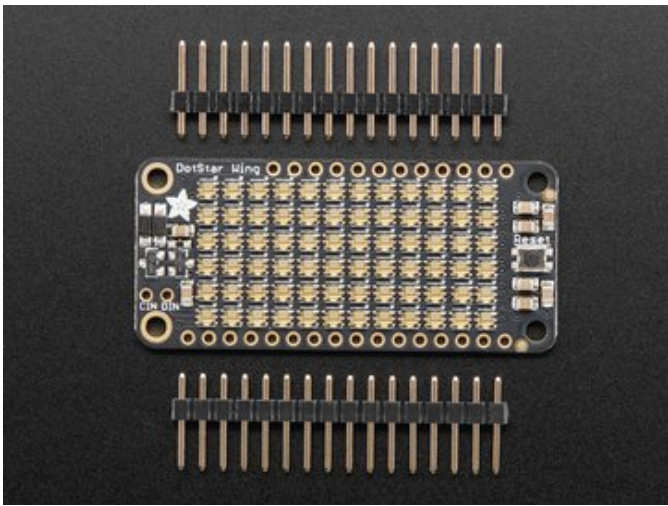
Rigid DotStar Matrices

A couple of small, non-flexible variants are available:



Adafruit DotStar High Density 8x8 Grid (<https://adafru.it/CIB>)

This is the tiniest little LED grid we could make, with **64** full RGB color pixels in a square that is only **1" by 1" square** (25.4mm x 25.4mm). Best of all, these are little DotStar LEDs, with built in PWM drivers, so you only need two digital I/O pins to get a-glowin'.

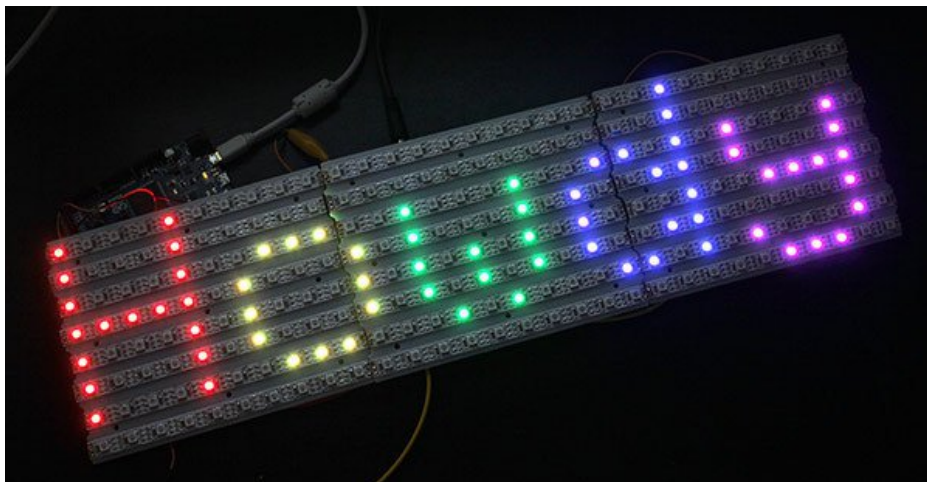


Adafruit DotStar FeatherWing (<https://adafru.it/AkQ>)

Stacks atop any of our **Feather** microcontroller boards. Arranged in a **6x12 matrix**, each 2mm by 2mm sized RGB pixel is individually addressable. Only two pins are required to control all the LEDs.

Finer Details About DotStar Matrices

As mentioned on the DotStar Strips page, keep power consumption in mind when working with DotStar matrices. With so many pixels at your disposal, it's easy to get carried away.



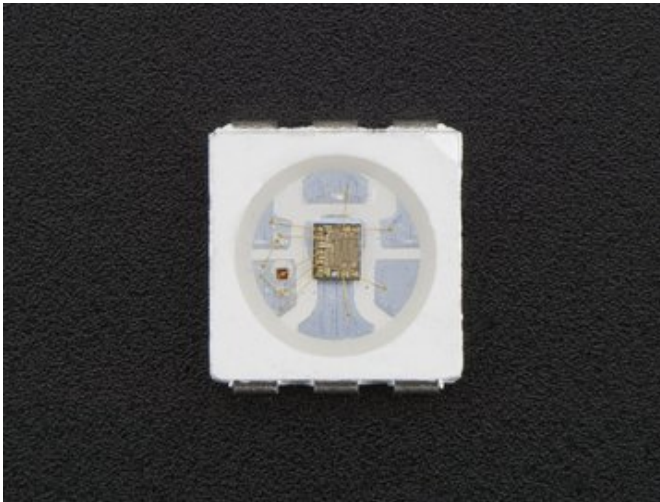
If you need a size or shape of DotStar matrix that's not offered here, it's possible to **create your own** using sections of **DotStar strip**!

DotStar matrices don't enforce any particular "topology" — some may have rows of pixels arranged left-to-right, others may alternate left-to-right and right-to-left rows, or they could be installed in vertical columns instead. **This will require some planning in your code.** Our *DotStarMatrix* library supports most matrix topologies.

Individual DotStars

For advanced users needing fully customized designs, **discrete DotStar** components are available. You'll need to provide your own PCB and surface-mount soldering skill.

SMT DotStars

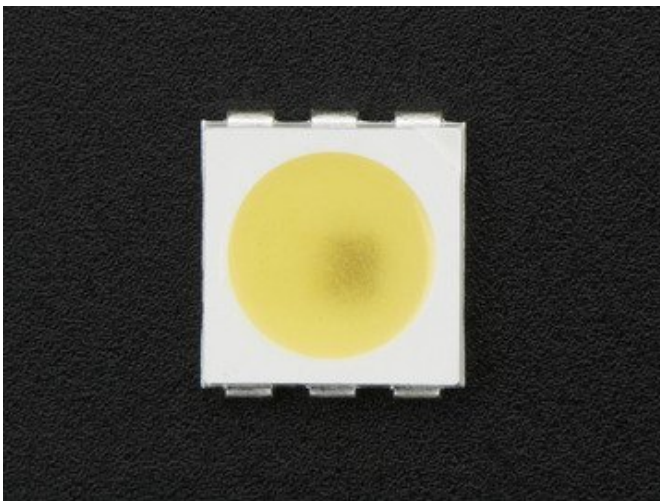


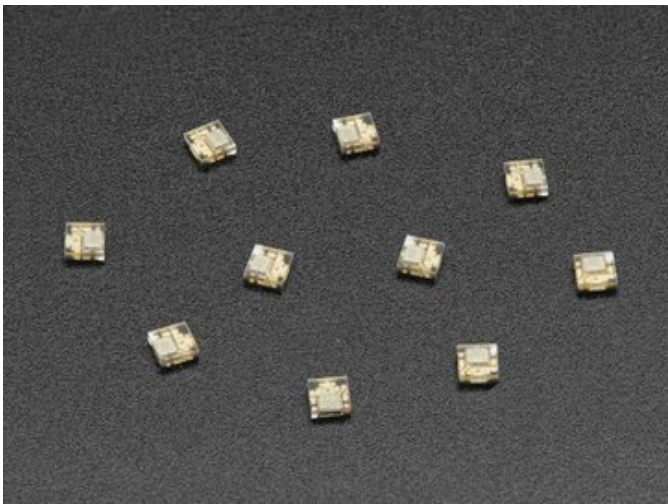
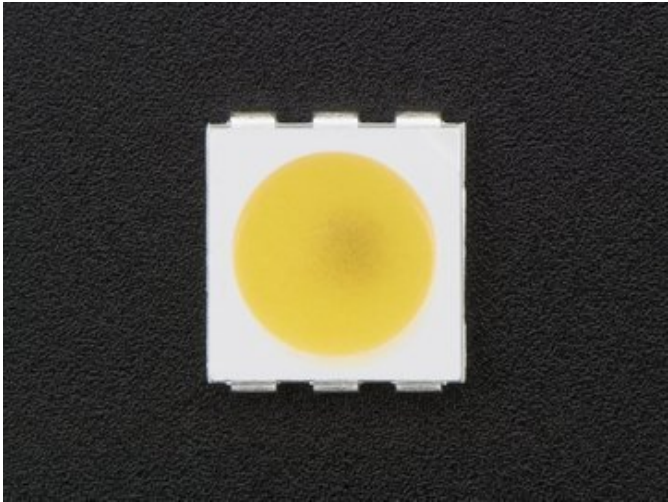
Surface-mount “5050” (5 millimeter square) DotStars are available in a few varieties:

- **5050 RGB LED — 10 Pack** (<https://adafru.it/CIC>)
- **5050 Cool White LED — 10 Pack** (<https://adafru.it/CID>)
- **5050 Warm White LED — 10 Pack** (<https://adafru.it/CIE>)

Approximate color temperature:

- Cool white: 6000–6500K
- Warm white: 3000K





Tiny surface-mount “2020” (2.0 millimeters square) DotStars are available in RGB (no pure white option):

- [DotStar Micro LEDs — 10 Pack](https://adafru.it/CIF) (<https://adafru.it/CIF>)

Power and Connections

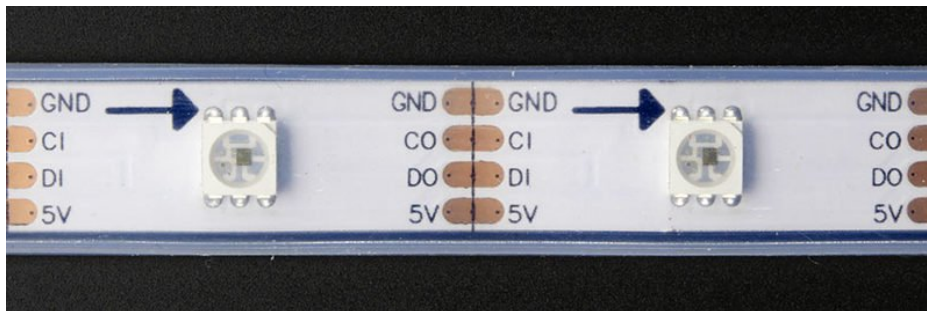
Powering DotStar LEDs

The power requirements for DotStars are pretty much identical to NeoPixels...in fact, we'll simply [refer you to the relevant page of the NeoPixel Überguide for pointers on estimating and routing power](https://adafru.it/iZe) (<https://adafru.it/iZe>). In summary:

- Estimate up to 60 milliamps peak for each pixel at full brightness white.
- A ground connection is required between the microcontroller and strip, in addition to the signal lines.
- For long strips, try to add a power tap every meter or so. This prevents a brown-out effect toward the end of the strip.
- As with NeoPixels, adding a 1000 μ F (6.3V or higher) capacitor close to the strip (between 5V and GND wires) is a good idea, do it!

Connecting DotStar LEDs

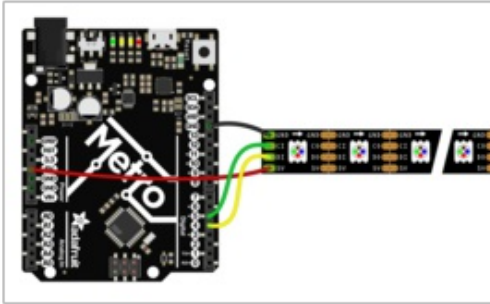
Our LED suppliers sometimes make unannounced production changes to the wiring. Therefore, **the best way to identify connections is a close visual examination of the strip.**



First, look for arrows printed along the strip next to each LED. These show the direction of data moving down the strip... your microcontroller connects at the originating (“in”) end, the arrows point toward the “out” end. (In the photo above, our microcontroller would be located off the left side.)

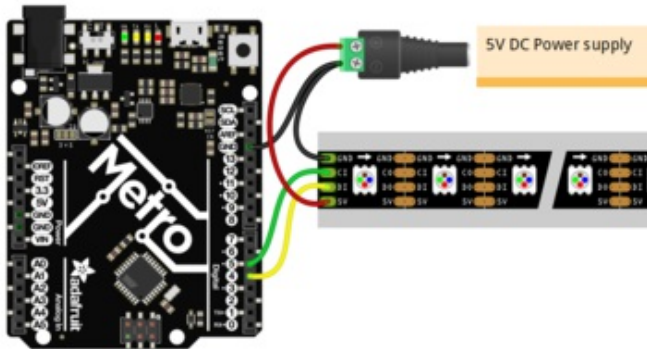
Second, look for labels on the strip to identify the function and order of the four wires: ground, 5 Volts, data and clock... usually labeled GND, 5V, D or DI (data input) and C or CI (clock input). These will go to corresponding pins on your microcontroller and power supply.

Again, due to production variations, you can't always count on wire colors or plug genders as a reliable indication of function, even if ordered at the same time. Take a close look to confirm before connecting anything.



The simplest wiring is if you can power the strip off of the microcontroller board itself. This is fine for small projects (one or two dozen DotStars max) because we don't light up a lot of LEDs at once.

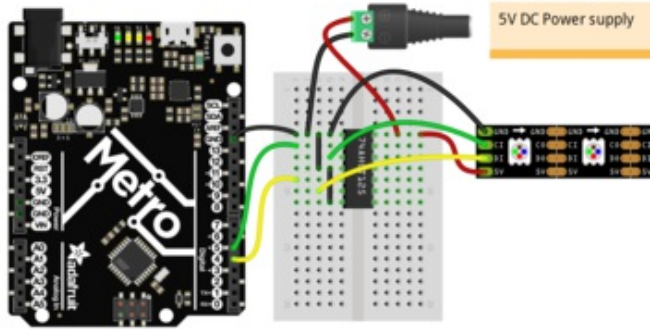
- Connect the strip **5V** pin to the board **5V**
- Connect the strip **GND** pin to board **GND**
- Connect the strip **CI** (Clock input) and **DI** (Data input) to the board's SPI **SCK** and **MOSI** pins (if using an **SPI** bus) or **any two digital pins** (if "bitbanging" the signals). We'll explain in more detail in the "**Software**" section of this guide.



For longer strips, when you need more than 1 Amp of current, you should power with an external 5V power adapter like so.

- Connect the strip **5V** pin to the power adapter **5V**
- Connect the strip **GND** pin to board **GND** and power adapter **GND**
- Connect the strip **CI** (Clock input) and **DI** (Data input) to two digital or SPI pins as explained above.

Important: **three** points are connected to **ground**: power supply, microcontroller and DotStar strip. If there's no common ground between the microcontroller and strip, the LED's won't function properly.



DotStars are **5 Volt** devices. They *may* respond to 3.3V signals, but this is not a guaranteed thing. If using a 3.3V controller (Feather, Raspberry Pi, etc.), add a **logic level shifter** to boost 3V logic to 5V...something like a [74AHCT125](https://adafru.it/e5g) (<https://adafru.it/e5g>) on the data and clock pins.

Software

DotStars got their start on Arduino, but have since branched out to other boards and languages.

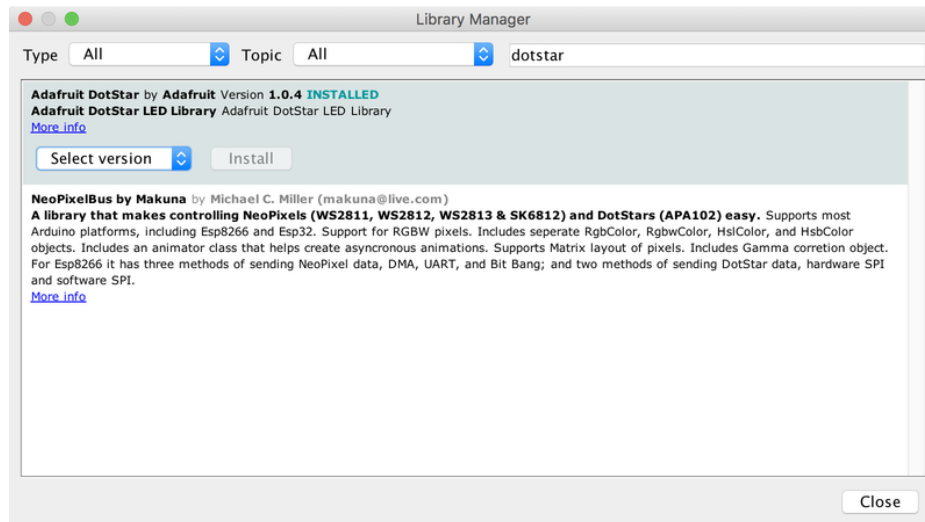
Pick a category from the left column for information specific to each coding environment.

Arduino Library Installation

Controlling DotStars “from scratch” is tedious, so we provide a library letting you focus on the fun and interesting bits. The library works with most mainstream Arduino boards and derivatives: Uno, Mega, Adafruit Feather, etc.

Install Adafruit_DotStar via Library Manager

Recent versions of the Arduino IDE (1.6.2 and later) make library installation super easy via the Library Manager interface. From the **Sketch** menu, > **Include Library** > **Manage Libraries...** In the text input box type in **"DotStar"**. Look for **"Adafruit DotStar by Adafruit"** and select the latest version by clicking on the popup menu next to the **Install** button. Then click on the **Install** button. After installation, you can click the **"close"** button.



Manually Install Adafruit_DotStar Library

If you're using an older version of the IDE, or just want to set things up manually, “classic” installation of the library is as follows: you can visit the [Adafruit_DotStar library page \(https://adafru.it/ej7\)](https://adafru.it/ej7) at Github and download from there, or just click this button:

<https://adafru.it/eio>

<https://adafru.it/eio>

1. Uncompress the ZIP file after it's finished downloading.
2. The resulting folder should contain the files **Adafruit_DotStar.cpp**, **Adafruit_DotStar.h** and an **“examples”** sub-folder. Sometimes in Windows you'll get an intermediate-level folder and need to move things around.
3. Rename the folder (containing the .cpp and .h files) to **Adafruit_DotStar** (with the underscore and everything), and place it alongside your other Arduino libraries, typically in your (home folder)/Documents/Arduino/Libraries folder. Libraries should *never* be installed in the “Libraries” folder alongside the Arduino application itself...put them in the subdirectory of your home folder!
4. Re-start the Arduino IDE if it's currently running.

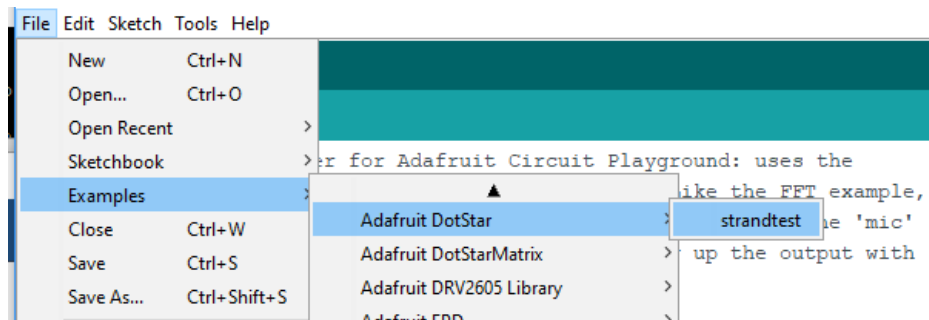
Here's a [tutorial \(https://adafru.it/aYM\)](https://adafru.it/aYM) that walks through the process of correctly installing Arduino libraries manually.

Another option for Arduino is the **FastLED** library (<https://adafru.it/eip>), featuring cutting-edge code with more color-handling and mathematical support functions. However, it's a little more tricky to use...so, if connecting DotStars for the first time, we ask that you **start with the Adafruit_DotStar library**. Once the hardware is confirmed working, you can

then graduate to whatever code or library you'd like!

A Simple Code Example: strandtest

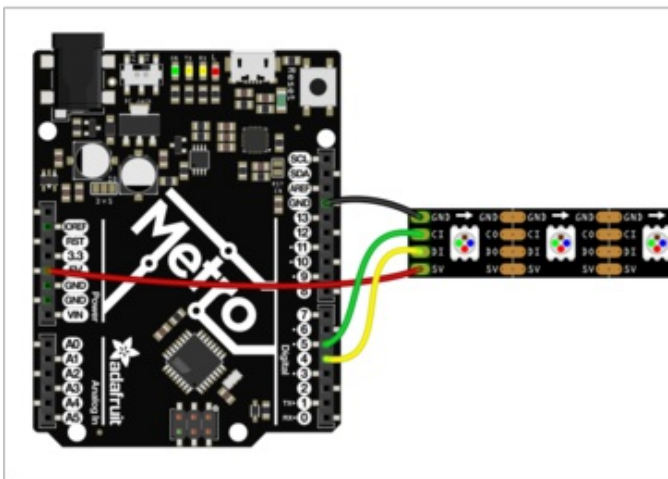
Launch the Arduino IDE. From the **File** menu, select **Sketchbook→Libraries→Adafruit_DotStar→strandtest**



(If the Adafruit_DotStar rollover menu is not present, the library has not been correctly installed, or the IDE needs to be restarted after installation. Check the installation steps above to confirm it's properly named and located.)

The “strandtest” example shows basic library use; declaring a strip object, setting LED colors and pushing this data to the strip via the `show()` method. For more advanced examples, **nearly any NeoPixel code should compile and run with DotStars, just changing the library `#include` and the strip declaration.**

Select your board type and serial port from the **Tools** menu, and try uploading to the board. If the DotStars are connected and powered as previously described, you should see a little light show.



If using a one-meter strip or less, it's usually OK to power off the microcontroller's 5V pin. Here's how it might look on an Arduino Uno or Adafruit Metro board:

- Connect the strip **5V** pin to the board **5V**
- Connect the strip **GND** pin to board **GND**
- Connect the strip **CI** (Clock input) to **digital pin 5**.
- Connect the strip **DI** (Data input) to **digital pin 4**.

For longer strips, see the “Power and Connections” page for guidance.

Some microcontroller boards won't have a pin 4 or 5. Not to worry, we'll show on the next page how to change the software for different connections.

Nothing happens!

Check your connections. The most common mistake is connecting to the output end of a strip rather than the input.

Something happens but the LEDs are blinking in a weird way!

99% of the time this is due to not having a shared ground wire connected to the Arduino. Make sure the **Ground** wire from the DotStars connects to BOTH your power supply ground AND the Arduino ground.

Arduino Library Use

It's assumed at this point that you have the `Adafruit_DotStar` library for Arduino installed and have run the `strandtest` example sketch successfully. If not, return to the prior page for directions to set that up.

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

All DotStar sketches begin by including the header file:

```
#include <Adafruit_DotStar.h>
```

Just below this in the **strandtest** example are some extra lines that are sometimes needed...

Most microcontrollers have some kind of **SPI** bus (a high-speed serial interface to devices). If so, there's an SPI-related header file that must be included. But a few microcontrollers (such as the diminutive Adafruit Gemma and Trinket) don't have SPI, in which case that line should be commented out (preceded with a `//`) or simply deleted.

Speaking of Gemma and Trinket...the next line, normally commented out, should be enabled (remove the `//`) if using one of those boards.

```
// Because conditional #includes don't work w/Arduino sketches...
#include <SPI.h>           // COMMENT OUT THIS LINE FOR GEMMA OR TRINKET
//#include <avr/power.h> // ENABLE THIS LINE FOR GEMMA OR TRINKET
```

The next few lines define the **length of the strip** (30 pixels in our example, but you can change this to more or less to match your setup) and **which pins** to use for the **data** and **clock** signals. The wiring diagrams previously shown had these on 4 and 5...but they can usually be any two pins, whatever works best for you and matches your wiring:

```
#define NUMPIXELS 30 // Number of LEDs in strip

// Here's how to control the LEDs from any two pins:
#define DATAPIN 4
#define CLOCKPIN 5
Adafruit_DotStar strip = Adafruit_DotStar(
  NUMPIXELS, DATAPIN, CLOCKPIN, DOTSTAR_BRG);
```

This is how we declare a DotStar *object*. We'll refer to this by name later to control the strip of pixels.

The last parameter is optional — this is the color data order of the DotStar strip, which has changed over time in different production runs. Default is `DOTSTAR_BRG`, so change this if you have an earlier strip. If the LEDs are basically working but coming up the wrong colors, this is usually the reason why...swap around the R, G and B until things look right.

If using an **SPI-capable** microcontroller, this usually provides better performance. However, this **must be wired to specific pins**, and it **varies among microcontrollers**. For example, on the Arduino Uno the SPI MOSI and SCK signals are on pins 11 and 13 respectively. You'll need a pinout diagram for other boards, look for the SPI **MOSI** and **SCK** pins!

The object declaration in this case is a little different...simply leave out the data and clock pin numbers:

```
Adafruit_DotStar strip = Adafruit_DotStar(NUMPIXELS, DOTSTAR_BRG);
```

Then, in the `setup()` function, call `begin()` to prepare the data and clock pins for DotStar 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 DotStars to an initial "off" state in case some were left lit by a prior program.

To set the color of an individual DotStar LED, use the `setPixelColor()` function, which can work a couple of different ways. Easiest usually involves passing a pixel number and red, green and blue color values:

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

`index` is the pixel number, starting at 0 (first pixel), then 1 (second pixel), up to the number of pixels in the strip minus one. Pixel indices outside this range will simply be ignored.

`red`, `green`, `blue` specify the color, each in a range from 0 (off) to 255 (maximum brightness).

For example, here's how we'd set the first pixel (index 0) to an orangey color (100% red, 50% green, 0% blue):

```
strip.setPixelColor(0, 255, 127, 0);
```

An alternate syntax takes just two arguments, a pixel index (same as before) and a single color value as a "packed" 24-bit integer (often in hexadecimal notation...something advanced programmers may be more comfortable working in):

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

You can "pack" 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.

`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 (only the least 24 bits are used).

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()` .

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 DotStar objects on different pins?

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

```
Adafruit_DotStar strip_a = Adafruit_DotStar(16, 3, 4);  
Adafruit_DotStar strip_b = Adafruit_DotStar(16, 5, 6);
```

The above declares two distinct DotStar objects, one with data and clock on pins 3 and 4, the other on pins 5 and 5. Each contains 16 pixels and is using the default color order.

Can I connect multiple DotStar strips to the same Arduino pins?

In many cases, yes. All the strips will then show exactly the same thing. This only works up to a point though...four strips wired to the same two pins is a good and reliable number. More than that starts to get "iffy."

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

Different versions of DotStar LEDs expect to receive color data in a different order...and occasionally it may change if it improves production efficiency or yield.

The last argument to the `Adafruit_DotStar()` constructor lets you try different color orders. Default if unspecified is `DOTSTAR_BRG` .

We *don't* change this in each release of the library because it's just an endless game of Whack-a-Mole...it's only a matter of time before the manufacturers use a different order again. Find what works with the hardware you have.

Most NeoPixel Code Adapts Easily to DotStars

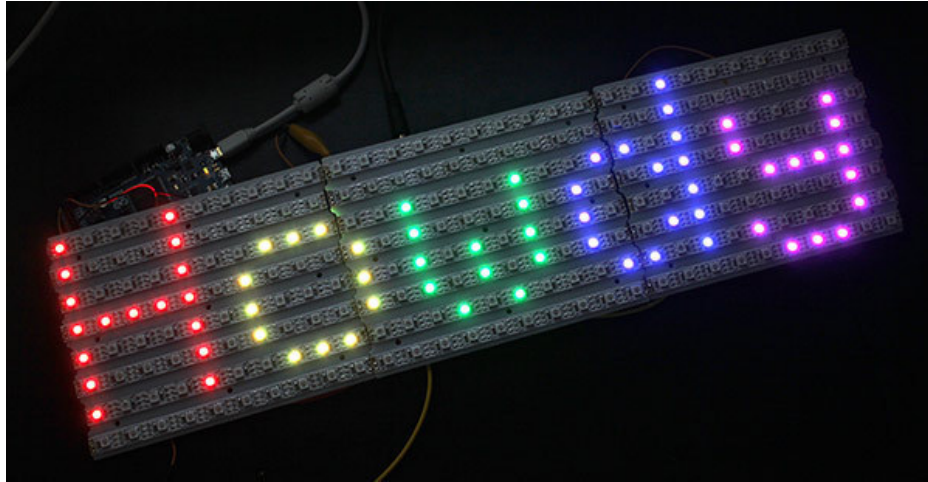
Nearly any NeoPixel code should compile and run with DotStars, just changing the library `#include` and the strip declaration...the remaining functions are roughly the same. There may be a few exceptions, but this is usually esoteric code that's doing NeoPixel-specific hardware trickery.

Pixels Gobble RAM

Each DotStar 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 one of our M0- or M4-equipped Metro Express or Feather boards. But if you're close and need just a little extra space, you can sometimes tweak your code to be more RAM-efficient. [This tutorial has some pointers on memory usage.](#) (<https://adafru.it/coj>)

DotStarMatrix Library



The `Adafruit_DotStarMatrix` library builds upon `Adafruit_DotStar` to create two-dimensional graphic displays using DotStar LEDs. You can then easily draw shapes, text and animation without having to calculate every X/Y pixel position. Small DotStar matrices are available in the shop. Larger displays can be formed using sections of DotStar strip, as shown above.

In addition to the `Adafruit_DotStar` library (which was already downloaded and installed in a prior step), `DotStarMatrix` requires **two** additional libraries:

1. [Adafruit_DotStarMatrix](https://adafru.it/iZA) (<https://adafru.it/iZA>)
2. [Adafruit_GFX](https://adafru.it/cBB) (<https://adafru.it/cBB>)

If you've previously used any Adafruit LCD or OLED displays, you might already have the latter library installed.

Installation for both is similar to `Adafruit_DotStar` before: using the **Arduino Library Manager** (in recent versions of the Arduino IDE) is recommended. Otherwise, if you manually download: unzip, make sure the folder name matches the .cpp and .h files within, then move to your Arduino libraries folder and restart the IDE.

Arduino sketches need to include all three headers just to use this library:

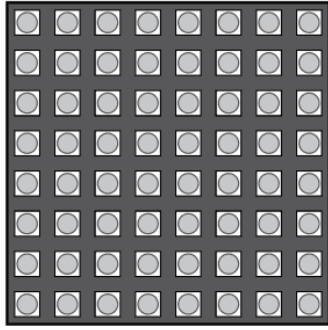
```
#include <Adafruit_GFX.h>
#include <Adafruit_DotStarMatrix.h>
#include <Adafruit_DotStar.h>
```

Layouts

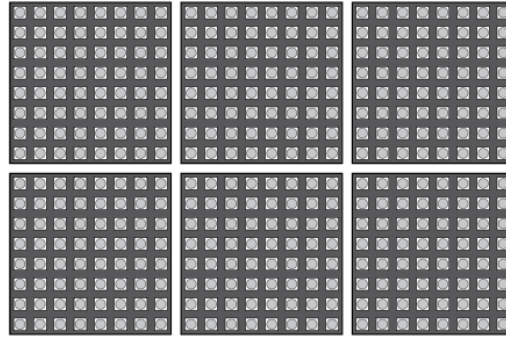
`Adafruit_DotStarMatrix` uses exactly the same coordinate system, color functions and graphics commands as the `Adafruit_GFX` library. If you're new to the latter, [a separate tutorial explains its use](https://adafru.it/aPe) (<https://adafru.it/aPe>). There are also example sketches included with the `Adafruit_DotStarMatrix` library.

We'll just focus on the *constructor* here — how to declare a two-dimensional display made from DotStars. Powering the beast is another matter, covered on the prior page.

The library handles both *single* matrices — all DotStars in a single uniform grid — and *tiled* matrices — multiple grids combined into a larger display:

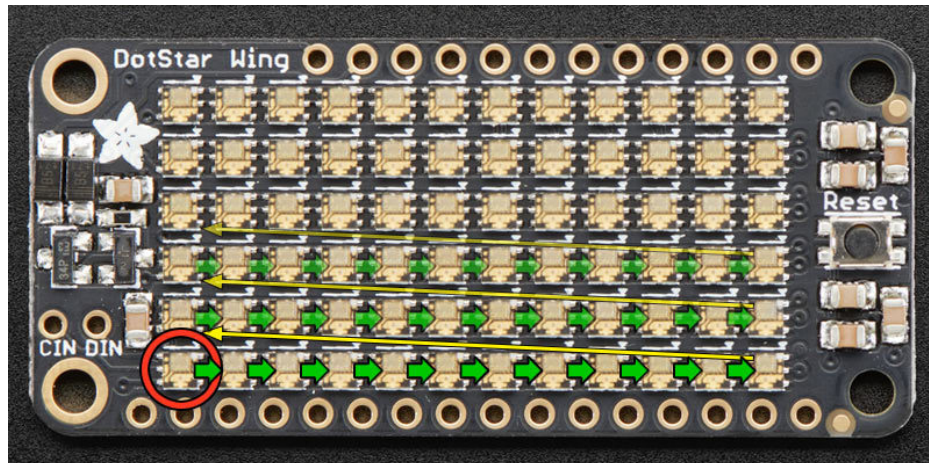


Single Matrix



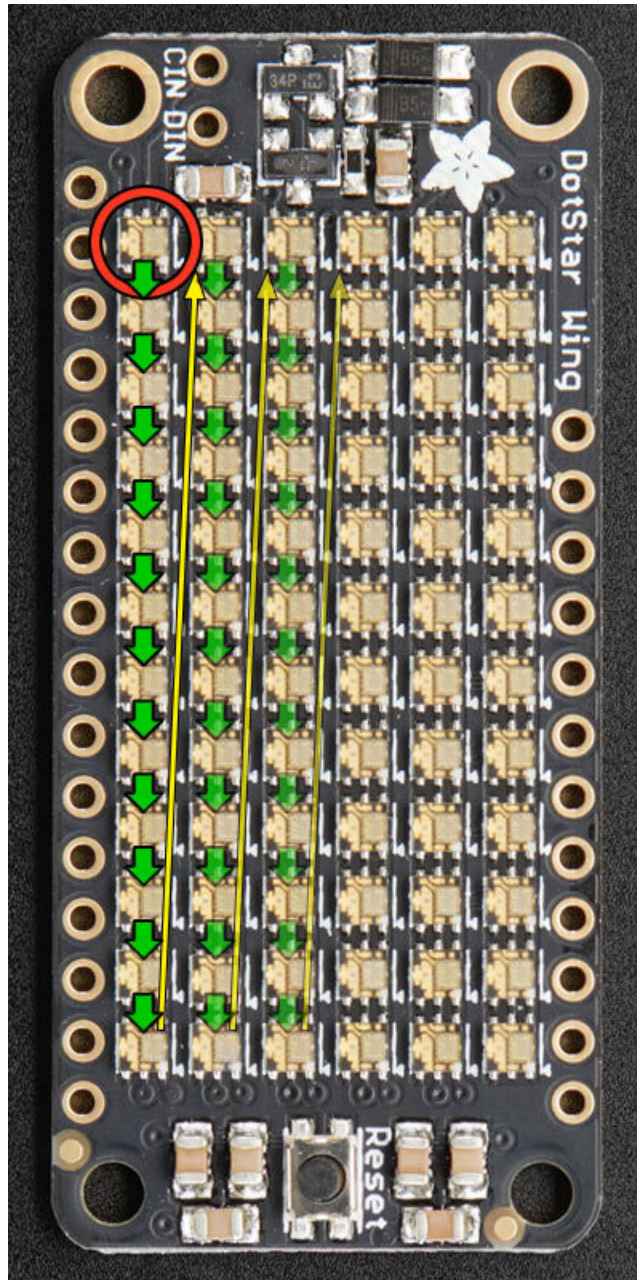
Tiled Matrices

Let's begin with the declaration for a single matrix, because it's simpler to explain. We'll be demonstrating the [Adafruit DotStar FeatherWing](https://adafruit.com/products/2471) (<https://adafruit.com/products/2471>) in this case — a 12x6 matrix of tiny DotStars. When looking at this FeatherWing with the text in a readable orientation, the first pixel, #0, is at the bottom left. Each successive pixel is right one position — pixel 1 is directly to the right of pixel 0, and so forth. At the end of each row, the next pixel is at the left side of the next row up. This isn't something we decide in code...it's how the DotStars are hard-wired in the circuit board comprising the shield.



We refer to this layout as *row major* and *progressive*. *Row major* means the pixels are arranged in horizontal lines (the opposite, in vertical lines, is *column major*). *Progressive* means each row proceeds in the same direction. Some matrices will reverse direction on each row, as it can be easier to wire that way. We call that a *zigzag* layout.

However...for this example, we want to use the FeatherWing in the “tall” direction, so the Feather board is standing up on the desk with the USB cable at the top. When we turn the board this way, the matrix layout changes...



Now the first pixel is at the **top left**. Pixels increment top-to-bottom — it's now **column major**. The order of the columns is still **progressive** though.

We declare the matrix thusly:

```
Adafruit_DotStarMatrix matrix = Adafruit_DotStarMatrix(
  6, 12, // Width, height
  11, 13, // Data pin, clock pin
  DS_MATRIX_TOP + DS_MATRIX_LEFT +
  DS_MATRIX_COLUMNS + DS_MATRIX_PROGRESSIVE,
  DOTSTAR_BGR);
```

The first two arguments — 12 and 6 — are the width and height of the matrix, in pixels. The next two arguments — 11 and 13 — are the pin numbers to which the DotStars are connected (data and clock, respectively). On the FeatherWing

this is hard-wired to digital pins 11 and 13, but on some Feather boards these physical pins have different numeric assignments, and standalone (non-FeatherWing) matrices are free to use other pins. See the `dotstar_wing.ino` example for pin assignments on other boards.

The next argument is the interesting one. This indicates where the first pixel in the matrix is positioned and the arrangement of rows or columns. The first pixel must be at one of the four corners; *which* corner is indicated by adding either `DS_MATRIX_TOP` or `DS_MATRIX_BOTTOM` to either `DS_MATRIX_LEFT` or `DS_MATRIX_RIGHT`. The row/column arrangement is indicated by further adding either `DS_MATRIX_COLUMNS` or `DS_MATRIX_ROWS` to either `DS_MATRIX_PROGRESSIVE` or `DS_MATRIX_ZIGZAG`. These values are all added to form a single value as in the above code.

```
DS_MATRIX_TOP + DS_MATRIX_LEFT +  
DS_MATRIX_COLUMNS + DS_MATRIX_PROGRESSIVE
```

The last argument is exactly the same as with the `DotStar` library, indicating the type of LED pixels being used. In some cases you can simply leave this argument off.

The point of this setup is that the rest of the sketch never needs to think about the layout of the matrix. Coordinate (0,0) for drawing graphics will always be at the top-left, regardless of the actual position of the first `DotStar`.

Why not just use the rotation feature in `Adafruit_GFX`?

`Adafruit_GFX` only handles rotation. Though it would handle our example above, it doesn't cover every permutation of rotation *and mirroring* that may occur with certain matrix layouts, not to mention the zig-zag capability, or this next bit...

Tiled Matrices

A *tiled* matrix is comprised of multiple smaller `DotStar` matrices. This is sometimes easier for assembly or for distributing power. All of the sub-matrices need to be the **same size**, and must be ordered in a predictable manner. The `Adafruit_DotStarMatrix()` constructor then receives some additional arguments:

```
Adafruit_DotStarMatrix matrix = Adafruit_DotStarMatrix(  
  matrixWidth, matrixHeight, tilesX, tilesY,  
  dataPin, clockPin, matrixType, ledType);
```

The first two arguments are the width and height, in pixels, of each tiled sub-matrix, not the entire display.

The next two arguments are the number of tiles, in the horizontal and vertical direction. The dimensions of the overall display then will always be a multiple of the sub-matrix dimensions.

The fifth and sixth arguments are the data and clock pin numbers, same as before and as with the `DotStar` library. The last argument also follows prior behaviors, and in many cases can be left off.

The second-to-last argument though...this gets complicated...

With a single matrix, there was a starting corner, a major axis (rows or columns) and a line sequence (progressive or zigzag). This is now doubled — similar information is needed both for the pixel order within the individual tiles, and the overall arrangement of tiles in the display. As before, we add up a list of symbols to produce a single argument describing the display format.

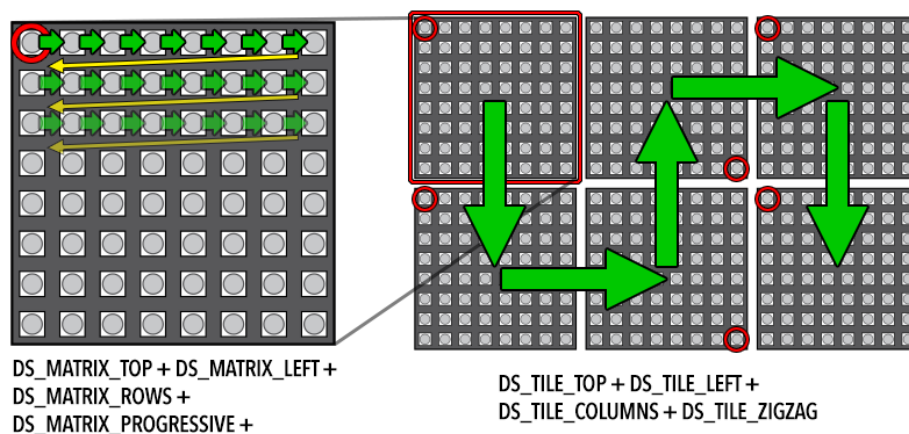
The `DS_MATRIX_*` symbols work the same as in the prior single-matrix case, and now refer to the individual sub-

matrices within the overall display. All tiles must follow the same format. An additional set of symbols work similarly to then describe the tile order.

The first tile must be located at one of the four corners. Add either DS_TILE_TOP or DS_TILE_BOTTOM and DS_TILE_LEFT or DS_TILE_RIGHT to indicate the position of the first tile. This is independent of the position of the first pixel within the tiles; they can be different corners.

Tiles can be arranged in horizontal rows or vertical columns. Again this is independent of the pixel order within the tiles. Add either DS_TILE_ROWS or DS_TILE_COLUMNS.

Finally, rows or columns of tiles may be arranged in progressive or zigzag order; that is, every row or column proceeds in the same order, or alternating rows/columns switch direction. Add either DS_TILE_PROGRESSIVE or DS_TILE_ZIGZAG to indicate the order. **BUT**...if DS_TILE_ZIGZAG order is selected, alternate lines of tiles must be rotated 180 degrees. This is intentional and by design; it keeps the tile-to-tile wiring more consistent and simple. This rotation is not required for DS_TILE_PROGRESSIVE.



Tiles don't need to be square! The above is just one possible layout. The display shown at the top of this page is three 10x8 tiles assembled from DotStar strip.

Once the matrix is defined, the remainder of the project is similar to Adafruit_DotStar. Remember to use `matrix.begin()` in the `setup()` function and `matrix.show()` to update the display after drawing. The `setBrightness()` function is also available. The library includes a couple of example sketches for reference.

Other Layouts

For any other cases that are not uniformly tiled, you can provide your own function to remap X/Y coordinates to DotStar strip indices. This function should accept two unsigned 16-bit arguments (pixel X, Y coordinates) and return an unsigned 16-bit value (corresponding strip index). The simplest row-major progressive function might resemble this:

```
uint16_t myRemapFn(uint16_t x, uint16_t y) {  
    return WIDTH * y + x;  
}
```

That's a crude example. Yours might be designed for pixels arranged in a spiral (easy wiring), or a Hilbert curve.

The function is then enabled using `setRemapFunction()`:

```
matrix.setRemapFunction(myRemapFn);
```

RAM Again

On a per-pixel basis, Adafruit_DotStarMatrix is no more memory-hungry than Adafruit_DotStar, requiring 3 bytes of RAM per pixel. But the number of pixels in a two-dimensional display takes off exponentially...a 16x16 display requires *four times* the memory of an 8x8 display, or about 768 bytes of RAM (nearly half the available space on an Arduino Uno). It can be anywhere from *tricky* to *impossible* to combine large displays with memory-hungry libraries such as SD or ffft. Fortunately 32-bit boards (e.g. Metro Express) are fairly mainstream now.

Gamma Correction

Because the Adafruit_GFX library was originally designed for LCDs (having limited color fidelity), it handles colors as 16-bit values (rather than the full 24 bits that DotStars are capable of). This is not the big loss it might seem. A quirk of human vision makes bright colors less discernible than dim ones. The Adafruit_DotStarMatrix library uses *gamma correction* to select brightness levels that are visually (though not numerically) equidistant. There are 32 levels for red and blue, 64 levels for green.

The `Color()` function performs the necessary conversion; you don't need to do any math. It accepts 8-bit red, green and blue values, and returns a gamma-corrected 16-bit color that can then be passed to other drawing functions.