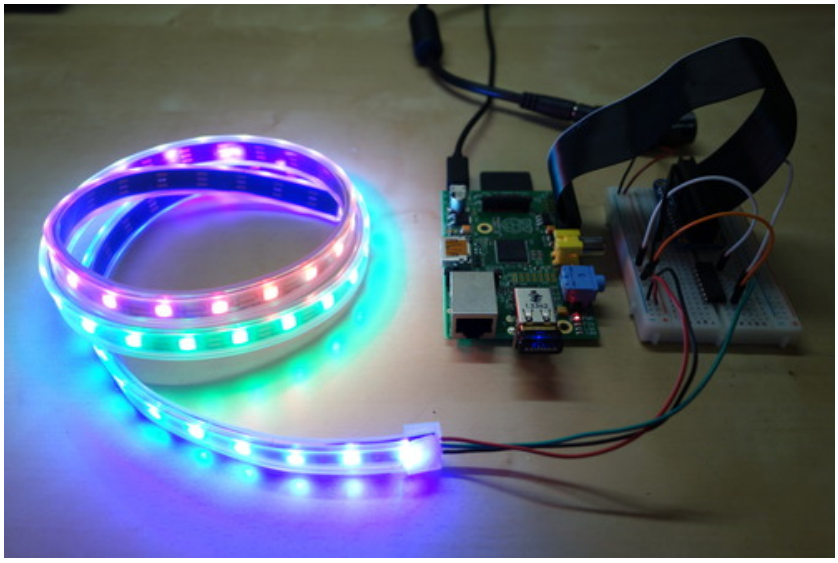
adafruit learning system

NeoPixels on Raspberry Pi

Created by Tony DiCola



Last updated on 2014-09-12 01:00:18 PM EDT

Guide Contents

Overview

Wouldn't it be fun to add bright, beautiful NeoPixels to your Raspberry Pi project? However NeoPixels, and the WS2811/2812 LEDs that make them up, require a data signal with very specific timing to work correctly. Because the Raspberry Pi runs a multi-tasking Linux operating system it doesn't have real-time control over its GPIO pins and can't easily drive NeoPixels. Typically a small microcontroller like a Trinket or Teensy can be used to communicate with the Raspberry Pi and generate the NeoPixel data signal. However thanks to the excellent rpi_ws281x (http://adafru.it/dYh) library created by Jeremy Garff (http://adafru.it/dYi), you can now control NeoPixels or WS2811/WS2812 LEDs directly from your Raspberry Pi!

Jeremy's library solves the real-time control problem by using the PWM and DMA hardware on the Raspberry Pi's processor. The PWM (pulse-width modulation) module can generate a signal with a specific duty cycle (http://adafru.it/dYj), for example to drive a servo or dim an LED. The DMA (direct memory access) module can transfer bytes of memory between parts of the processor without using the CPU. By using DMA to send a specific sequence of bytes to the PWM module, the NeoPixel data signal (http://adafru.it/dYk) can be generated without being interrupted by the Raspberry Pi's operating system.

The great thing about Jeremy's library is that it does all the hard work of setting up PWM and DMA to drive NeoPixels. On top of rpi_ws281x I've added a small Python wrapper which makes the library look and feel like the Arduino NeoPixel library. This guide will show you how to install the rpi_ws281x library and Python wrapper so you can use NeoPixels in your own Raspberry Pi projects!

Before you get started you will want to be familiar with how to connect to a Raspberry Pi's terminal using SSH (http://adafru.it/dYl). It will also be helpful to check out the NeoPixel Uberguide (http://adafru.it/dYm) for more information on using NeoPixels.

Wiring

Wiring NeoPixels to work with a Raspberry Pi is quite simple. The only issue to deal with is converting the Pi's GPIO from 3.3V up to about 5V for the NeoPixel to read. There are two ways you can do this level conversion, either with a simple 1N4001 power diode or with a level converter chip like the 74AHCT125.

Note that you *might* be able to get your NeoPixels to work without any level conversion, but it's not really guaranteed because the data line needs to be at least 0.7 * VDD (5 volts), or about 3.5 volts. Try one of the level conversion options below if you can't directly drive the pixels from your Raspberry Pi.

The diode method is a quick way to reduce the power supply voltage slightly so the NeoPixels can read the Pi's 3.3V output. However you need to be careful to use a diode that can handle all the current drawn by the NeoPixels. The diodes Adafruit sells only handle 1 Amp of continuous current so they're good for driving up to about 16 NeoPixels at full 100% bright white - and about 50 NeoPixels if they're all lit with various colors. Also because the NeoPixels aren't running at 5 volts they might be a little dimmer than normal.

A level converter chip like the 74AHCT125 is a better method because it will convert the Pi's 3.3V output up to 5V without limiting the power drawn by the NeoPixels. You'll get full NeoPixel brightness that's only limited by the current capability of the power supply.

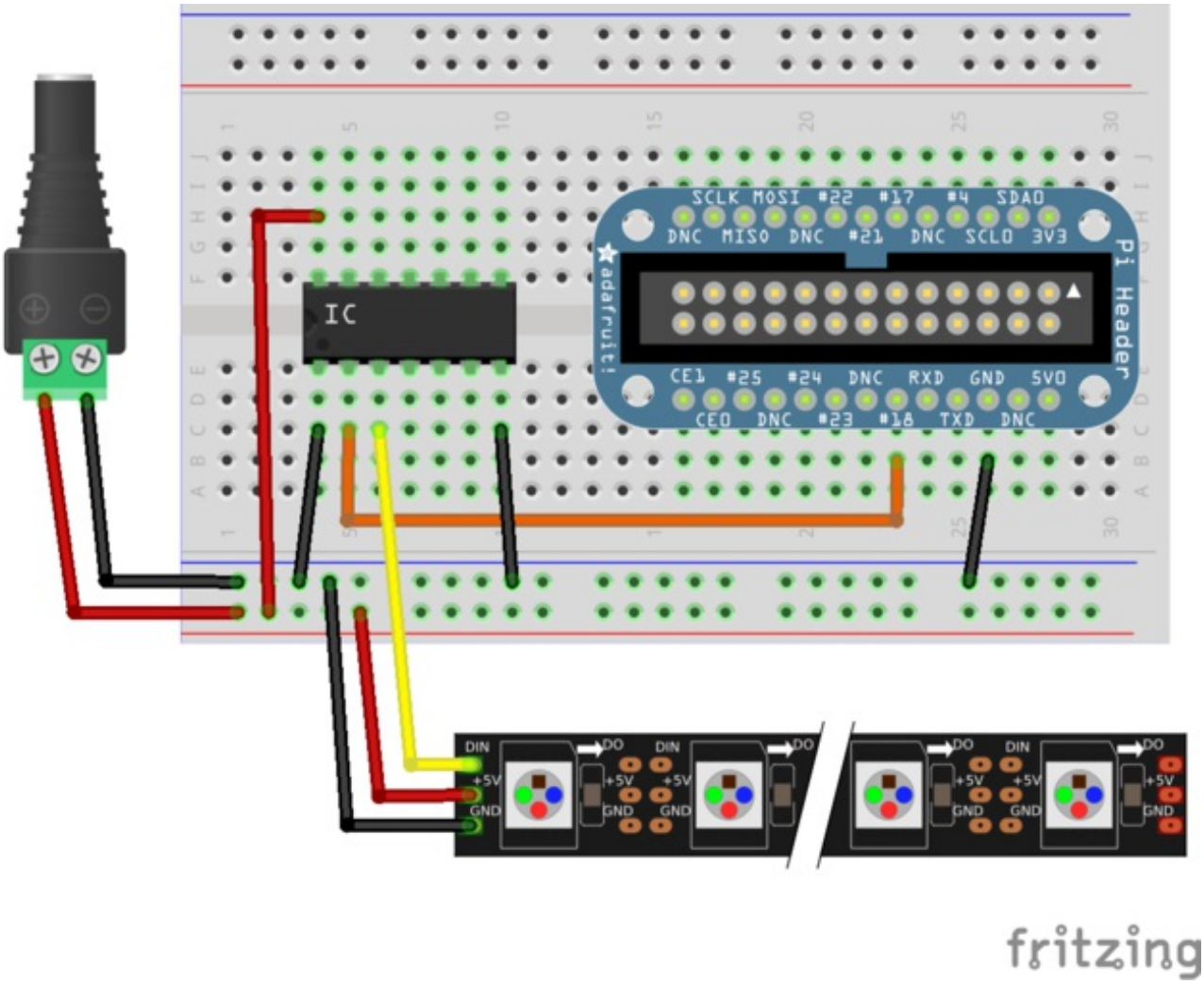
To follow this guide you'll need the following parts:

- 5 volt power supply (http://adafru.it/276)
    - Make sure the supply can handle powering all the NeoPixels. Remember each pixel can draw up to 60 mA so don't skimp on the power supply!
- 2.1mm female barrel jack adapter (http://adafru.it/368)
- Solderless breadboard (http://adafru.it/64) and hookup wires (http://adafru.it/758)
- 1N4001 diode (http://adafru.it/755) **OR** 74AHCT125 level converter (http://adafru.it/1787)
- NeoPixels! (http://adafru.it/dYn)
    - Note that this method of driving NeoPixels is limited by the maximum size of a kernel memory page and can in theory control up to about **450 LEDs.**
    - Also be aware the rpi_ws281x library only supports **GRB** NeoPixels/WS281x LEDs right now. Support for RGB and BGR will likely come in the future (but pull requests are happily accepted!).

Do not power NeoPixels from the Raspberry Pi's 5V output! The Pi cannot source enough current to light many pixels and will be damaged. Use a good quality external 5V power supply that can handle the current demands of all the pixels.

Level-converter Chip Wiring

If you're using a 74AHCT125 level converter, wire up your hardware as follows:

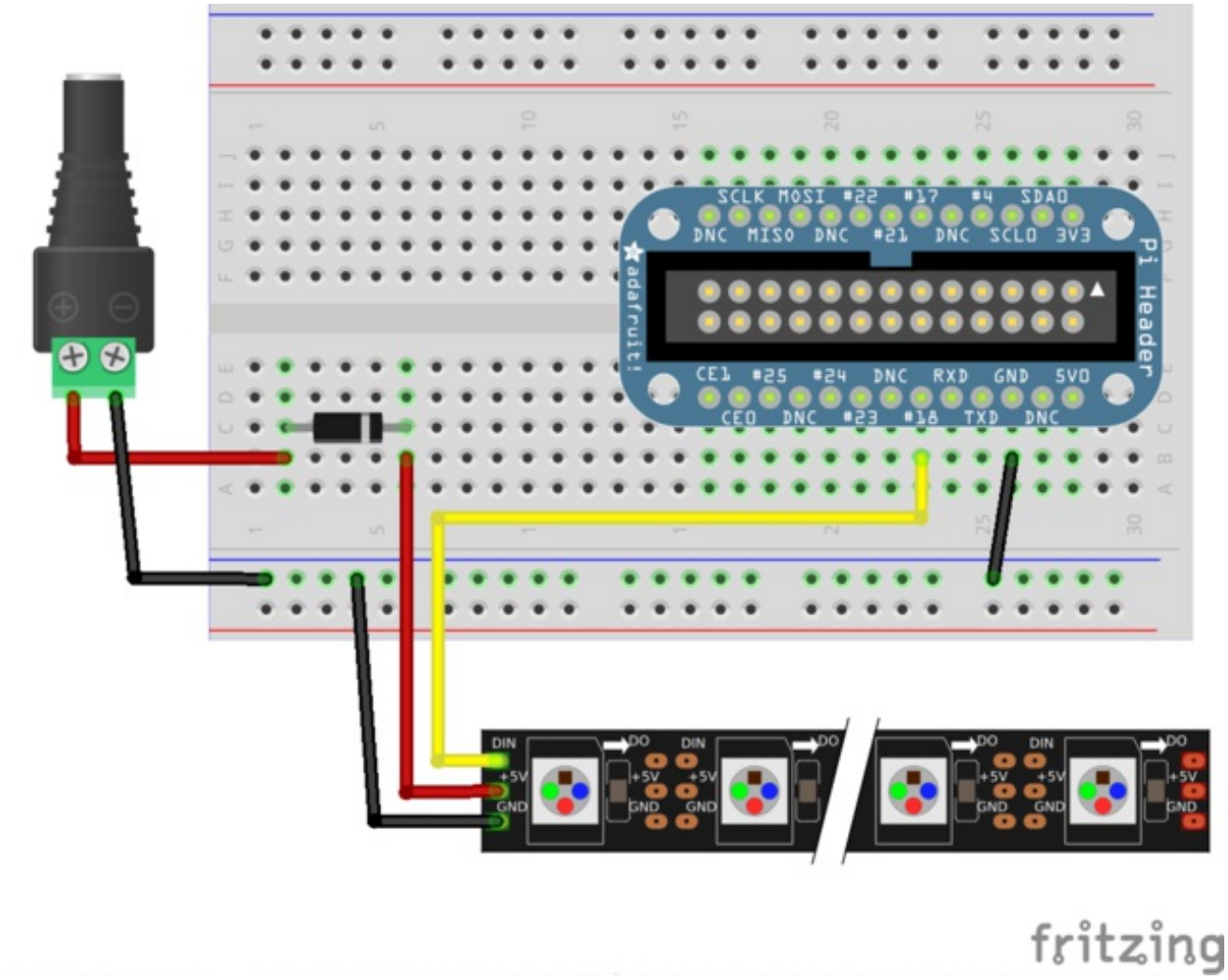


- Connect **power supply ground** to **74AHCT125 ground & 1OE** pins, **Raspberry Pi ground**, and **NeoPixel ground**.
- Connect **power supply 5V** to **74AHCT125 VCC** and **NeoPixel 5V**.
- Connect **Raspberry Pi pin 18** to **74AHCT125 pin 1A**.
- Connect **74AHCT125 pin 1Y** to **NeoPixl DIN**.

If you aren't sure where the pins are on the 74AHCT125 chip then check its datasheet (http://adafru.it/dYo) for more information.

Diode Wiring

If you're using a 1N4001 diode wire up your hardware as follows:



- Connect **power supply ground** to **Raspberry Pi ground**, and **NeoPixel ground**.
- Connect **power supply 5V** to **1N4001 diode anode** (side *without* the stripe).
- Connect **1N4001 diode cathode** (side *with* the stripe) to **NeoPixel 5V**.
- Connect **Raspberry Pi pin 18** to **NeoPixel DIN**.

Make sure to get the orientation of the diode correct, with the cathode (side with the stripe) going to the NeoPixel!

Software

The rpi_ws281x library (http://adafru.it/dYh) is the key that makes using NeoPixels with the Raspberry Pi possible. In this guide I'll use a fork of the library (http://adafru.it/dYp) I created to add a Python wrapper around it, however in the future this fork will likely be integrated into the main library.

Easy Installation

To start using the library quickly you can install a pre-made binary distribution. Note that this only works on the Raspbian operating system with Python 2.7. If you're using a different operating system or Python version you will need to skip down to the steps to compile the library.

Connect to your Raspberry Pi in a terminal and execute the following commands:

```
wget https://github.com/tdicola/rpi_ws281x/raw/master/python/dist/rpi_ws281x-1.0.0-py2.7-linux-armv
sudo easy_install rpi_ws281x-1.0.0-py2.7-linux-armv6l.egg
```

After running the easy_install command you should see a response such as the following:

```
Processing rpi_ws281x-1.0.0-py2.7-linux-armv6l.egg
Copying rpi_ws281x-1.0.0-py2.7-linux-armv6l.egg to /usr/local/lib/python2.7/dist-packages
Adding rpi-ws281x 1.0.0 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/rpi_ws281x-1.0.0-py2.7-linux-armv6l.egg
Processing dependencies for rpi-ws281x==1.0.0
Finished processing dependencies for rpi-ws281x==1.0.0
```

Now download an example of using the library by executing:

```
wget https://github.com/tdicola/rpi_ws281x/raw/master/python/examples/strandtest.py
```

That's it! You've successfully installed the library and downloaded an example. You can skip down to the Strandtest Example section of this page to learn more about running the example and using the library.

Compile & Install rpi_ws281x Library

These steps will show you how to compile the rpi_ws281x library and install a Python wrapper around it. This is only necessary if you couldn't use the easy installation steps above.

To start, connect to a terminal on the Raspberry Pi and execute the following commands to install some dependencies:

```
sudo apt-get update
sudo apt-get install build-essential python-dev git scons swig
```

Now run these commands to download the library source (actually a fork of the library) and compile it:

```
git clone https://github.com/tdicola/rpi_ws281x.git
cd rpi_ws281x
scons
```

After running the scons command above you should see the library successfully compiled. Next you can install the Python library by executing:

```
cd python
sudo python setup.py install
```

After running those commands the Python wrapper around the rpi_ws281x library should be generated and installed.

Inside the examples subdirectory you can find a couple examples of using the library. **strandtest.py** is an example of using the high-level Python library which looks like the Arduino NeoPixel library. **lowlevel.py** is an example of using the lower level SWIG-generated wrapper around the rpi_ws281x library. You probably don't want to use the low level API unless you're writing your own library--stick to using the higher level API in **strandtest.py**!

Strandtest Example

To demonstrate the usage of the NeoPixel Python wrapper I'll walk through the code for the **strandtest.py** example. First to run example make sure your hardware is setup as described on the previous page. Then open **standtest.py** in a text editor and scroll down to the section of code that configures the LEDs here:

```
# LED strip configuration:
LED_COUNT   = 16      # Number of LED pixels.
LED_PIN     = 18      # GPIO pin connected to the pixels (must support PWM!).
LED_FREQ_HZ = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA     = 5       # DMA channel to use for generating signal (try 5)
```

```
LED_INVERT  = False   # True to invert the signal (when using NPN transistor level shift)
```

Change the value of LED_COUNT to be the number of LEDs in your NeoPixel strand/board/ring. The rest of the settings don't need to be changed, but they're good to review in case you change the hardware in the future.

Save the file, and then run it by executing:

```
sudo python strandtest.py
```

Make sure to run the script as root by using the sudo command. The rpi_ws281x library has to access the Pi hardware at a low level and requires running as root!

It might take a few seconds for the program to initialize and then you should see the NeoPixels light up and animate in different color wipes, theater lights, and rainbow animations. Press **Ctrl-C** at any time to quit the example. If you see an error about glibc detected double free or corruption you can ignore it (when the program is exited with Ctrl-C it can sometimes abruptly kill the process before it can gracefully clean-up its memory).

To understand how to use the high level Python wrapper open **strandtest.py** in a text editor again and follow along below.

```
import time

from neopixel import *
```

At the top of the script are the module imports. In this example the Python standard time module is included to access its sleep function. More importantly though the next line imports all the functions from the **neopixel** module. This **neopixel** module defines the high-level wrapper around the rpi_ws281x library.

```
# LED strip configuration:
LED_COUNT   = 16      # Number of LED pixels.
LED_PIN     = 18      # GPIO pin connected to the pixels (must support PWM!).
LED_FREQ_HZ = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA     = 5       # DMA channel to use for generating signal (try 5)
LED_INVERT  = False   # True to invert the signal (when using NPN transistor level shift)
```

The next lines are what you saw earlier to configure the LEDs. In particular make sure the count and pin are set to what you're using with your hardware. Be careful with the pin value because it **must** be a pin that supports hardware PWM on the Pi, like pin 18.

The remaining configuration generally doesn't need to change, but for reference you can control the frequency of the NeoPixel control signal (typically 800khz, but sometimes 400khz), DMA channel (try 5, but there are 15 channels with values 0-14 available), and a boolean to invert the control signal. You might need to invert the control signal if you're using a NPN transistor as a level-converter (not described in this guide).

```
# Define functions which animate LEDs in various ways.
def colorWipe(strip, color, wait_ms=50):
 """Wipe color across display a pixel at a time."""
 for i in range(strip.numPixels()):
  strip.setPixelColor(i, color)
  strip.show()
  time.sleep(wait_ms/1000.0)
```

Next a few functions are defined to run the animations in the example. These functions take in an **Adafruit_NeoPixel** object (defined later in the code) as their first parameter and call functions on that object to set LED colors. In particular you can see these functions are used on the **Adafruit_NeoPixel** object:

- **numPixels()** - This function returns the number of pixels in the LED strip/matrix/ring/etc. This is handy when looping through all the pixels to animate or change them in some way.
- **setPixelColor(pos, color)** - This function sets the LED pixel at position pos to the provided color. Color should be a 24-bit value where the upper 8 bits are the red value, middle 8 bits are the green value, and lower 8 bits are the blue value. You'll actually see a little later in the sketch a helper function that lets you define a color with just these red, green, blue component values.
- **setPixelColorRGB(pos, red, green, blue)** - Although not shown in this example, you can call this function to set the color of a pixel directly using the specified red, green, and blue component values. Each component value should be a number from 0-255 where 0 is the lowest intensity and 255 is the highest intensity.
- **show()** - This function is very important because it's the only function that will actually change the color of the LEDs. After you've set pixel colors you must call show() to update the hardware!

I'll skip describing the rest of the animation functions since they're pretty similar in their usage of the NeoPixel library.

```
# Create NeoPixel object with appropriate configuration.
strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT)
# Intialize the library (must be called once before other functions).
strip.begin()
```

Further down in the example code you'll find the lines above which create the

Adafruit_NeoPixel object. The initializer function for this object takes in all the parameters defined earlier like the number of pixels, GPIO pin connected to the pixels, etc.

Notice that after creating the **Adafruit_NeoPixel** object the **begin()** function is called. It's very important to call **begin()** once before you make other calls on the **Adafruit_NeoPixel** object!

```
print 'Press Ctrl-C to quit.'
while True:
 # Color wipe animations.
 colorWipe(strip, Color(255, 0, 0))  # Red wipe
 colorWipe(strip, Color(0, 255, 0))  # Blue wipe
 colorWipe(strip, Color(0, 0, 255))  # Green wipe
 # Theater chase animations.
 theaterChase(strip, Color(127, 127, 127))  # White theater chase
 theaterChase(strip, Color(127,   0,   0))  # Red theater chase
 theaterChase(strip, Color(  0,   0, 127))  # Blue theater chase
 # Rainbow animations.
 rainbow(strip)
 rainbowCycle(strip)
 theaterChaseRainbow(strip)
```

The final part of the example enters a loop where it calls the animation functions defined earlier. The important thing to notice here is how a color is defined with the **Color()** function. This function takes 3 parameters, the red, green, and blue component values of the color. Each component is a value from 0-255 where 0 is the lowest intensity and 255 is the highest intensity.

For example to generate the brightest red color possible you would call **Color(255, 0, 0)**. Or to generate a moderately pink color you could call **Color(128, 0, 128)**.

That's all there is to using the NeoPixel Python wrapper around the rpi_ws281x library!

If you'd like to see all the functions available in the library, run the command:

```
pydoc neopixel
```

For reference here's the full output of the pydoc command:

```
Help on module neopixel:

NAME
    neopixel
```

```
DESCRIPTION
    # Adafruit NeoPixel library port to the rpi_ws281x library.
    # Author: Tony DiCola (tony@tonydicola.com)

CLASSES
    __builtin__.object
        Adafruit_NeoPixel

    class Adafruit_NeoPixel(__builtin__.object)
     |  Methods defined here:
     |
     |  __del__(self)
     |
     |  __init__(self, num, pin, freq_hz=800000, dma=5, invert=False)
     |      Class to represent a NeoPixel/WS281x LED display.  Num should be the
     |      number of pixels in the display, and pin should be the GPIO pin connected
     |      to the display signal line (must be a PWM pin like 18!).  Optional
     |      parameters are freq, the frequency of the display signal in hertz (default
     |      800khz), dma, the DMA channel to use (default 5), and invert, a boolean
     |      specifying if the signal line should be inverted (default False).
     |
     |  begin(self)
     |      Initialize library, must be called once before other functions are
     |      called.
     |
     |  getPixelColor(self, n)
     |      Get the 24-bit RGB color value for the LED at position n.
     |
     |  getPixels(self)
     |      Return an object which allows access to the LED display data as if
     |      it were a sequence of 24-bit RGB values.
     |
     |  numPixels(self)
     |      Return the number of pixels in the display.
     |
     |  setBrightness(self, brightness)
     |      Scale each LED in the buffer by the provided brightness.  A brightness
     |      of 0 is the darkest and 255 is the brightest.  Note that scaling can have
     |      quantization issues (i.e. blowing out to white or black) if used repeatedly!
     |
     |  setPixelColor(self, n, color)
     |      Set LED at position n to the provided 24-bit color value (in RGB order).
     |
     |  setPixelColorRGB(self, n, red, green, blue)
     |      Set LED at position n to the provided red, green, and blue color.
     |      Each color component should be a value from 0 to 255 (where 0 is the
```

```
    |      lowest intensity and 255 is the highest intensity).
    |
    |  show(self)
    |      Update the display with the data from the LED buffer.
    |

FUNCTIONS
    Color(red, green, blue)
        Convert the provided red, green, blue color to a 24-bit color value.
        Each color component should be a value 0-255 where 0 is the lowest intensity
        and 255 is the highest intensity.
```

If you run into issues with the library, try asking for help on the Adafruit forums (http://adafru.it/dYq) or even open a bug on the library's home on github (http://adafru.it/dYh).