Wondering what’s next for npm? Check out our public roadmap! »

# rpi-ws281x-native

0.10.1 • Public • Published 2 months ago

Readme

Explore BETA

1 Dependency

13 Dependents

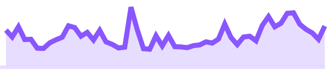
20 Versions

Install

```
> npm i rpi-ws281x-native
```

Weekly Downloads

123



| Version | License |
|---|---|
| 0.10.1 | MIT |

| Unpacked Size | Total Files |
|---|---|
| 217 kB | 59 |

| Issues | Pull Requests |
|---|---|
| 25 | 2 |

Homepage

github.com/beyondscreen/node-rpi-ws281x-native

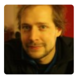
Repository

github.com/beyondscreen/node-rpi-ws281x-native

Last publish

2 months ago

Collaborators



Try on RunKit

Report malware

# control ws281x-LEDs with node.js

> if you happen to know C++ and node/iojs/V8, I would really appreciate any help and feedback on this module. There is certainly lots of room for improvement.

This module provides native bindings to the rpi_ws281x library by Jeremy Garff to provide a very basic set of functions to write data to a strip of ws2811/ws2812 LEDs. **this will only run on the Raspberry Pi.**

## setup

this module is available via npm:

```
npm install rpi-ws281x-native
```

if you prefer installing from source:

```
npm install -g node-gyp
git clone --recursive https://github.com/beyondscreen/node-rpi-ws281x-native.git
cd rpi-ws281x-native
npm install
node-gyp rebuild
```

## basic usage

this module mainly exports four functions to send data to the LED-String.

```
exports = {
    /**
     * configures PWM and DMA for sending data to the LEDs.
     *
     * @param {Number} numLeds  number of LEDs to be controlled
     * @param {?Object} options  (acutally only tested with default-values)
     *                           intialization-options for the library
     *                           (PWM frequency, DMA channel, GPIO, Brightness)
     */
    init: function(numLeds, options) {},

    /**
     * register a mapping to manipulate array-indices within the
     * data-array before rendering.
     *
     * @param {Array.<Number>} map  the mapping, indexed by destination.
     */
    setIndexMapping: function(map) {},

    /**
     * set the overall-brightness for the entire strip.
     * This is a fixed scaling applied by the driver when
     * data is sent to the strip
     *
     * @param {Number} brightness the brightness, value from 0 to 255.
     */
    setBrightness: function(brightness) {},

    /**
     * send data to the LED-strip.
     *
     * @param {Uint32Array} data  the pixel-data, 24bit per pixel in
     *                            RGB-format (0xff0000 is red).
     */
    render: function(data) {},

    /**
     * clears all LEDs, resets the PWM and DMA-parts and deallocates
     * all internal structures.
     */
    reset: function() {}
};
```

```
};
```

### Johnny-Five Compatability

To use this with Johnny-Five and Raspi-io you will need to tell Raspi-io to exclude the data pins that the LEDs are using. Otherwise the pins will not be available and you won't be able to control the LEDs.

For instance, if your LEDs is connected to GPIO18 of the Pi:

```
const ws281x = require('rpi-ws281x-native');
const five = require('johnny-five');
const Raspi = require('raspi-io');

const board = new five.Board({
  io: new Raspi({ excludePins: 'GPIO18'}) // Exclude GPIO18 from raspi-io
});
```

### Index-Mapping

As the wiring of the LEDs not neccessarily corresponds to the pixel-ordering in the data-array, this module supports index-remapping. So, if you are building a grid of LEDs you can just use an alternating, top-to-bottom or mirrored wiring and use the remapping in order to use a unified structure in the incoming data-arrays.

### Events

In addition to that, the exported object is an `EventEmitter` that will emit the following Events:

- `beforeRender` : emitted just before the data is prepared and sent to the LED-driver. The handler will receive the pixel-data array (an `Uint32Array` ) as single argument. As this event is handled synchronously, you can use this to manipulate the data before it is sent to the LED-Strip.
- `render` : emitted after the data has been sent to the LED-Strip. The single argument passed to the handler is the final pixel-data array, after index-remapping and gamma-correction.

## testing basic functionality

connect the WS2812-strip to the raspberry-pi as described **here** and run the command `sudo node examples/rainbow.js <numLeds>` . You should now see some rainbow-colors animation on the LED-strip.

## needs to run as root

As the native part of this module needs to directly interface with the physical memory of the raspberry-pi (which is required in order to configure the PWM and DMA-modules), it always has to run with root-privileges (there are probably ways around this requirement, but that doesn't change the fact that the node-process running the LEDs needs access to the raw physical memory – a thing you should never allow to any user other than root).

If you are using this module as part of a program that should not be run with elevated privileges, it would be a good idea to have the LED-driver running in a seperate process. In such a case you could use the openpixelcontrol protocol to send the pixel-data to the driver-process. A stream-based node-implementation and some more information **can be found here**.

## Hardware

There is a guide **over at adafruit.com** on how to get the hardware up and running. I followed these instructions by the word and had a working LED-strip.

Essentially, you need the Raspberry Pi, a logic-level converter to shift the output-voltage of the GPIO from 3.3V up to 5V (the guide mentions the 74AHCT125, mine is an 74HCT125N which works just as well) and of course a LED-Strip or other types of WS2812-LEDs.

To connect all that together, I'd recommend buying a small breadboard and some jumper-wires. Also, consider buying a 5V power-supply that can deliver up to 60mA per LED (so you'll need up to 6A (30W) to fully power 100 LEDs). For smaller applications, a decent USB-charger should do.

### Buying stuff

A short checklist of what you will need:

- Raspberry-PI and SD-Card
- 5V power-supply (Meanwell for instance builds really good ones)
- LED-Strip with WS2811/WS2812 Controllers (there are several other controller-variations that are not supported)
- a breadboard and some jumper-wires (m/m as well as at least two f/m to connect the GPIO-Pins)
- a 3.3V to 5V logic-level converter (74AHCT125 or 74HCT125N, others will probably also work)
- more wire to connect the LED-strips

You can buy everything at **adafruit.com**, **sparkfun**, on ebay or your favourite electronics retailer (germany: check **conrad electronic**, **watterott** or **reichelt** where i bought most of my stuff).

## Known Issues

There is a conflict where the internal soundcard uses the same GPIO / DMA / PWM functions that are needed to run the LED-drivers. As far as I know you can not use both at the same time.

To disable audio, comment out the following line in config.txt contained on the boot partion.

#dtparam=audio=on

Default is off.

### Keywords

**raspberry** **ws2811** **ws2812** **led** **hardware**