



# PixelPusher Hardware Configuration Guide

If you have bought Heroic Robotics FastPixel LED strip, use the [HR Strip Quick Start Guide](#)

If you have bought Heroic Robotics BrightPixel LED pixels, use the [HR Two Inch Pixels Quick Start Guide](#) or the [HR Single LED Pixels Quick Start Guide](#).

If you have bought Heroic Robotics FastPixel Rigid Strip Modules, use the [HR Rigid Strip Quick Start Guide](#).

If you have bought Heroic Robotics BetterPixel products, use the [HR BetterPixel Quick Start Guide](#).

If you want to use PixelPusher to control servo motors, see the document [Motion Control with PixelPusher](#).

Otherwise, here is the full documentation for all supported PixelPusher config options.

## How to Push Pixels

-or-

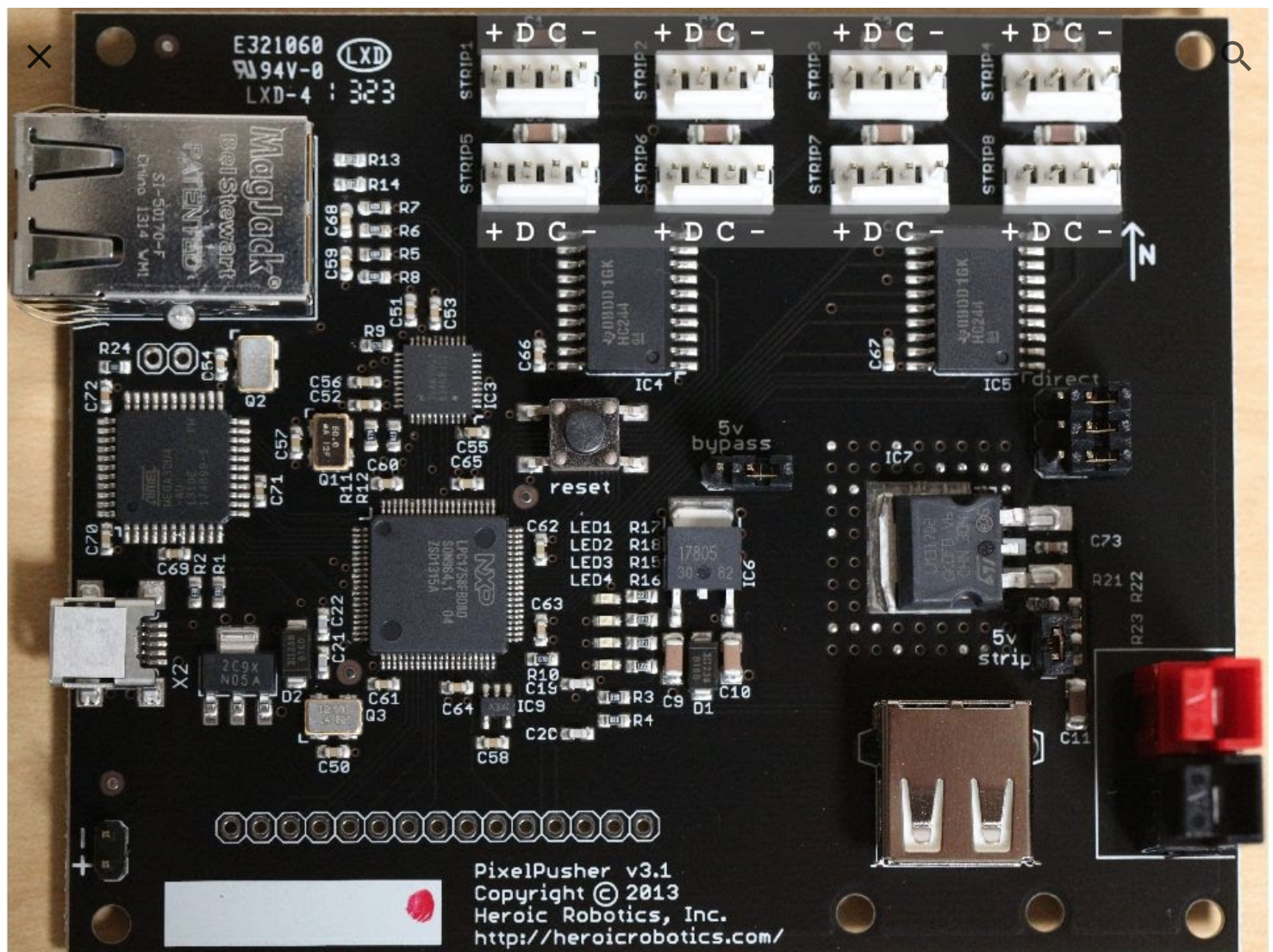
## A concise guide to the care and feeding of the Heroic Robotics PixelPusher

*document version 0.8*

Hi! Thanks for buying a PixelPusher. You're awesome. So now you have this gadget and you want to know how to set it up. Well, we can help you with that.

First, let's get oriented. This is the layout of the PixelPusher v3.





In the top right hand corner you can see a compass. This is so that we can use compass directions in this manual. To the west of the compass, you can see eight connectors marked X1 to X8. These are where the strips (or strings of pixels) are connected. We'll just call them strips for convenience. From the picture above, you can determine which pins correspond to positive voltage (+), data (D), clock (C), and ground (-).

At the western edge of the board, there's an ethernet jack, and in the southwest and southeast corners are power connectors. Just to the west of the big power connector, there's a USB host port. North of this power connector there are a bunch of jumpers, which you will need to set before you connect power.

Let's talk a little more about power. There are two power connectors- one is an Anderson PowerPole connector block, and is where you can feed big honking currents in. The other is a 0.1" pin header, suitable only for much smaller currents. It's provided for convenience, as it mates with many LiPo batteries- but be warned that the PixelPusher contains no undervoltage protection, so it is possible to damage your battery this way.

The red terminal of the Anderson PowerPole connector should connect to the positive output from the power supply.

PixelPusher itself supports supply voltages between 4.5 and 30 volts.



There are three jumpers on the v3 board. They must be set correctly before you can apply voltage to the board. They are

marked thusly:

✕  
■ 5v strip



- Selects 5.1 volt output for the onboard regulator (as opposed to 10.6v)
- 5v bypass
  - Connects the microcontroller module directly to input power, rather than using the regulator.
  - **WARNING** - if your power supply is below 7 volts, you must set the 5v bypass to the 'on' position. If your power supply is greater than 7 volts, you must set the 5v bypass jumper to the 'off' position.
- Direct
  - Connects the strip bus directly to the power input, rather than using the regulator. **WARNING** - You **must** enable this if you want to drive large numbers of pixels. The onboard regulator is only capable of 2 amps.

The reason for this is that there are many different types of strip and pixel, and they need different voltages. PixelPusher supports several.

If you are using strips with a supply voltage the same as your power supply's output, you should set the direct jumper to the 'on' position. Otherwise, beware! it is possible to damage strips by connecting them to a voltage that is too high.


If you are using 5v strips and the onboard regulator, you must set the "5v strip" jumper. Otherwise, the output voltage from the onboard regulator will be 10.63 volts, which is correct for many '12 volt' strips. This voltage is low enough to allow SD600A based strips to completely turn off their red pixels. If you are using LPD8806 strips, or the WS2801 LED pixels, then you can set the "direct" jumper to the closed position and bypass the regulator.

Finally, if you are using the direct jumper, you should also set the Reg disc jumper to the 'off' position. Otherwise, it must be set to the 'on' position.

*Advanced users note: Pixels with other power requirements than these must either be powered externally, or both the Reg disc and direct jumpers must be set to 'off' and power fed into the TP1 pads from an external source. The negative pad of TP1 is the eastern one, and is connected to the system's groundplane.*

Now that you have set all these jumpers correctly, you may apply power to the system. The PowerPole connector is recommended, and is rated for up to 40 amps; if you intend to use high currents such as these it is recommended that you bridge across the "direct" jumper with solder, or place a wire-ended fuse in its position. Heroic Robotics can perform this modification before shipping if necessary. The V3 boards have much meatier construction and shouldn't need this modification anyway.

Connect a standard ethernet cable to the Ethernet jack and plug its other end into a switch or router.

PixelPusher does not support auto-MDIX and as a consequence tends not to work well with a direct connection to some laptop computers. When power is applied you should see a blue light come on atop the microcontroller module, followed a few  seconds later by orange and green LEDs on the ethernet jack. (Both ethernet LEDs are green on the v3 PixelPusher.) Your switch should show a 100 Mbit connection, with activity

once a second as the PixelPusher announces itself. This is a successful test! If it doesn't work, check your power supply and ethernet wiring.



## Configuration

Now, to configure the device. PixelPusher v2 can read its configuration from two locations; a flash drive on the microcontroller module, or a USB memory device attached to the USB host port in the southeast corner of the board. The PixelPusher v3 can use a USB memory device, or a configuration stored in EEPROM using the PixelPusher Config Tool. We recommend using a USB memory device, because it makes PixelPushers interchangeable and easily replaceable during servicing- just pull the module and swap the flash device. There are two holes in the PCB near the USB connector which are intended to allow a zip tie to be passed around a USB memory stick in order to retain it in high vibration environments.

It reads its configuration once, at startup, a fraction of a second after power is applied or reset is pressed. If it loses link on its ethernet port for fifteen continuous seconds, it will reboot itself and re-read the configuration, so this is a way to reset a device remotely- just pull its ethernet cable and wait a few seconds. If the data watchdog is enabled, then PixelPusher will reboot if it does not receive any data for the programmed interval.

The configuration file, whether placed on USB memory or internally to the microcontroller module, is named "pixel.rc" and must be in the root directory of the device; it is a plain text file- NOT an RTF- and either DOS or UNIX line endings are supported. You can edit it with any text editor such as Notepad or TextEdit. On the Mac, we recommend that you use TextWrangler, as TextEdit is prone to saving RTF format files even when you ask it not to.

The USB memory device must be formatted with a FAT filesystem- neither NTFS, HFS nor ExFAT are supported- and we recommend getting small, cheap flash memory devices, since the config files are small. We use SanDisk Cruzer Fit memory sticks in the 8 Gbyte size.

If PixelPusher does not find a configuration file, or necessary options are missing from the one it has, it will default to using DHCP to configure the network, driving eight strips based on the LPD8806 chip, with 240 pixels each. These are the correct settings for the strips from Illumination Supply that we recommend- most users will not need to use a configuration file at all.

If you are using the Illumination Supply 2" LED pixels or the single LED pixels, then the correct settings for early units are as follows. Note that later runs of pixels do not require the swap line. For the single pixels, the correct colour ordering is grb, not rgb as shown below.

pixels=24

swap=12345678

strip1=ws2801

order1=rgb





strip2=ws2801



order2=rgb

strip3=ws2801

order3=rgb

strip4=ws2801

order4=rgb

strip5=ws2801

order5=rgb

strip6=ws2801

order6=rgb

strip7=ws2801

order7=rgb

strip8=ws2801

order8=rgb

## Writing a pixel.rc file

The pixel.rc file consists of a list of options in the form:

name=value

which can be placed in any order, and interspersed with blank lines and comments in the form:

# this is a comment

so that you can remember which device this was intended for, or which strip is which.

The supported options are listed below. Note that these options are case sensitive and do not work if you use capital letters in place of lower case!

### Network related options:

#### ether, netmask and gateway:

ether=10.2.3.4

netmask=255.255.255.0



gateway=10.2.3.1



This sets the IP address of the device, and disables DHCP. If ether is present, netmask and gateway must also be specified. They have the same format.

#### **dhcp\_timeout:**

```
dhcp_timeout=12
```

This controls how long the PixelPusher will wait for a DHCP response, and it's measured in tries. Each try takes five seconds, and a DHCP request is sent at the start of each, so dhcp\_timeout=12 will try twelve times and will take a maximum of one minute. This is useful if you're using a router that is slow to boot up. When the dhcp timeout counter expires, PixelPusher will default to an AutoIP address in the 169.254.x.x space.

#### **announcetarget:**

```
announcetarget=192.168.1.222
```

PixelPusher announces its presence once a second by a broadcast message to all hosts on the local subnet; however, broadcasts are not routable, so if you want to run the PixelPusher across a router, you need to be able to specify a host to send the announcements to. If this configuration option is present, PixelPusher will send a copy of the announcement message to the specified host as well as broadcasting.

#### **announce\_from:**

```
announce_from=192.186.1.27
```

If you are using the targetted announcements (see announcetarget above) then you will probably also want to change the address that the announcements come from so that they may be routed appropriately. This option allows you to change the address announced in the targetted announcement packet (the broadcast announcement always includes the local IP address, since if you are receiving the broadcast announcement you must be on the same subnet.)

#### **port:**

```
port=1138
```

Sets the UDP port on which PixelPusher listens for traffic. This setting is reflected in the announcement messages, so no configuration on the computer side is required.

#### **artnet\_universe and artnet\_channel:**

```
artnet_universe=1
```

```
artnet_channel=7
```

While PixelPusher does not natively understand Art-Net, there is a bridge application that converts Art-Net and streaming ACN traffic to drive PixelPushers. In order that the system be stateless and

repeatable, while maintaining the dynamic, self-organizing features of PixelPusher, each PixelPusher has a configuration variable for its starting Art-Net universe and channel. This is reported to the bridge application, and it maps the pusher's pixels into Art-Net space starting at that position, filling universes until it gets to the end. Note that universe and channel 0 are reserved for internal use.

#### **stripsperpacket:**

stripsperpacket=2

Normally the PixelPusher firmware computes how many strips will fit in a standard 1500 octet datagram based on the network configuration; however, you may want to reduce this value if you have a non-standard network, so we provide this option. Note that increasing this value above the normal maximum may cause problems.

#### **group:**

group=2

Sets which group of PixelPushers this device belongs to.

#### **controller:**

controller=45

Sets the PixelPusher's position in its group.

#### **Signalling related options:**

PixelPusher supports many different types of strip, and new strip types are being added regularly. If your chosen driver chip is not supported, or doesn't seem to work, contact [support@heroicrobotics.com](mailto:support@heroicrobotics.com) and ask us what's up.

Some of these options are global, and some are per-strip. These are the global ones:

#### **servo\_interval:**

servo\_interval=20000

When using the servo output feature of PixelPusher Firmware 1.32 and later, this sets the time interval in microseconds between servo pulses.

#### **servo\_minimum:**

servo\_minimum=1000



This sets the minimum duration of a servo pulse, in microseconds. As per specification, 1000 microseconds is

the shortest pulse, but your servos may need different settings.



#### **blank\_strips\_on\_idle:**

```
blank_strips_on_idle=1
```

This option causes PixelPusher to blank its strips if it hasn't received any data in a while. The value it uses for a particular strip is set using the `startn` option, described below.

#### **servo\_multiplier:**

```
servo_multiplier=4
```

This sets how long the pulse will be for a given position output. The unit is microseconds per pixel value tick. Since pixel values range from 0 to 255, a `servo_multiplier` of 4 will give a pulse that ranges from 1000 to 2022 microseconds, for a `servo_minimum` of 1000.

#### **update:**

```
update=1
```

PixelPusher normally updates strips with data as soon as it arrived over the network. However, this can mean that tens of milliseconds elapse between updates on a slow network. To avoid this, the `update` option uses a strip update bitmap to only update the strips once a complete set has been received. Note that this is not currently supported on SD600A strips; they always update immediately.

#### **stripsattached:**

```
stripsattached=3
```

Sets the number of strips attached. The lowest-numbered ports are always used.

#### **turbo\_mode:**

```
turbo_mode=16
```

The special turbo mode allows ultra-high-framerate control of up to two LPD8806 strips. The number sets the signalling rate in MHz. Only two strips are supported, and they require a special wiring configuration (each strip has two plugs). With this mode it is possible to drive 240 pixel strips at rates exceeding 1000 frames per second. Only the LPD8806 is supported; other strip architectures currently supported do not PWM fast enough for this mode to be worthwhile. Note that at these high frame rates, PixelPusher may consume a lot of network bandwidth! Note that the Early Access PixelPusher does not support turbo mode.

#### **pixels:**

```
pixels=60
```



Sets the number of pixels in each strip. **Note** that sending extra pixels (ie., more than the strip physically has)



generally has no effect, so it's okay to mix strip lengths; the network code does not support variable lengths of strips on a single PixelPusher. The longest strip supported is 480 pixels, giving each PixelPusher the ability to support a total of 3840 pixels. Note that some strip types will override this option. In this case, any "extra" pixels configured will be ignored. For "wide pixels" strips, you must configure twice as many software pixels as there are hardware pixels.

**swap:**

swap=23456

This is a little more technical. Most strips have a data line and a clock line. If you look at the board layout image at the start of this manual, you'll see that the pins on the strip connectors are labelled "+ D C -". D stands for Data and C stands for Clock. Some strips do not use this orientation; if data and clock lines need to be swapped, that's what the swap option is for. It takes a list of strip numbers, and swaps the data and clock lines of those strips in software. In this way, it is possible to support most strips without twisted cables.

You should note that the ws2801 pixels we sell require a swap, and some of the earlier lpd8806 strips do too. If you can't get them to work properly, try swapping them.

**copy:**

copy=2345678

This option copies data from strip 1 to the listed strips. It allows for large areas to be lit at low bandwidth cost, but otherwise is probably best replaced with some software option. It's useful for testing, anyway, so we've left it in.

**group:**

group=27

This option tells the PixelPusher which group number it is a member of. This can be any whole number. The PixelPusher library uses these numbers to organise PixelPusher cards into different arrays- if you have several different displays on the same network, for example, you can use the group option to distinguish the PixelPushers in one display from another. The default group is 0. When using the Java library, `registry.getStrips()` will return a list of all strips in all groups; `registry.getStrips(27)` will return a list of all the strips attached to controllers in group 27.

**controller:**

controller=17

This option tells the PixelPusher its position in its group. PixelPushers are ordered by ascending group, and then by ascending controller number, and then by the MAC address of each controller. The default controller number is zero, so by default PixelPushers are ordered by MAC address.

**ws28delay:**

ws28delay=7



When driving LED strips or pixels that use the WS2801 controller chip, some strips may require slower signalling than usual. This option allows the signals to be slowed down slightly from the standard rate, which allows longer cables and higher-power pixels to be used. The default is 0. Each increment of 1 adds approximately 0.5 microseconds per bit. Note that (obviously!) if you reduce the rate at which signals are sent, the LEDs will update more slowly, but (less obviously) you may be able to run the signals further between the LEDs and the PixelPusher.

### onewire:

onewire=2

Strips that use the WS2811 chipset are available in two speeds. The slower speed of 400 kHz is the default; this value multiplies the frequency, so to use 800 kHz signalling you should set this option to 2. Note that the ws2811 is slow to update (five times slower than the other supported strips, even at 800 kHz), and not recommended for current designs due to its poor signal integrity.

### Per-strip options

There are four per-strip options. These are they:

**stripn:** where *n* is the strip number

strip1=lpd8806

strip2=sd600a

strip3=lpd8806

strip4=ws2801

strip5=ws2811

strip6=lpd6803

strip4=mbi6030

strip3=servo

strip7=p9813

The stripn option selects which of the supported driver chips is used in that strip. Currently, MBI6030, LPD8806, SD600A, WS2801, P9813, LPD6803, TLC59711, APA102 and WS2811 chips (and chips with similar signalling to these) are supported, in any combination, and the default is LPD8806. The WS2811 and LPD6803 are not recommended for new designs. LPD8806 is used in FastPixel products, WS2801 is used in BrightPixel products, and APA102 is used in BetterPixel products.

A special setting for a strip is "servo", which converts that port into a PPM servo drive compatible with radio control type servos. In this mode, the first pixel's first component controls the servo whose signal line is connected to the data pin, and the second pixel's first component controls the servo whose signal line is

connected to the clock pin of the relevant port.



**flagn:** where  $n$  is the strip number

flag1=1

flag2=0

flag3=1

The flagn option allows the PixelPusher to report strip-specific layout data to the controlling software. There is currently only one supported flag bit, which is bit 1: this specifies that the strip attached to this port is a Heroic Robotics/Illumination Supply High CRI lighting strip. The default flag value is zero.

**startn:** where  $n$  is the strip number

start1=ffffff

start2=a55555

start3=101010

start4=cafe21

The startn option takes hex values for the colour with which to initialize your pixels. It will write this colour to the pixels very shortly after the power is connected- within a small fraction of a second- and this is useful in order to provide a "failsafe" where the lights come on even if the network is down.

**ordern:** where  $n$  is the strip number

order1=gbr

order2=rgb

order3=bgr

order4=bgr

The ordern option selects which colour order should be used for pixels in a given strip. The default is the same as "rbg", but note that the actual wire order of pixels pushed out on the wire may differ according to manufacturer datasheet recommendations. If you find that your strips are giving the wrong colours, try changing up the order.

**Miscellaneous options:**

**canned\_file**

canned\_file=1



The `canned_file` option, if set to 1, disables the network stack entirely and instead plays a pre-recorded show from a file named `canned.dat` stored on the USB stick. This allows PixelPusher to be used standalone, without any controlling computer or mobile device. You can generate these files using the Java library- see the forum for more information on how to do this. The maximum file size is four gigabytes, which allows for several days of uninterrupted playback.

If the option is set to 2, the new format with absolute timing is used. This format is emitted by versions of the library released after 20140806, and provides much more accurate playback speed.

### **data\_watchdog\_time:**

`data_watchdog_time=180`

This option enables the data watchdog. If no data is received within the time specified (in seconds), the PixelPusher will reboot itself. The default is zero, which disables the data watchdog.

## When Things Go Wrong

Sometimes things don't work out the way you expect them to. When you're dealing with a device like PixelPusher, it can be difficult to figure out what's going on. Fortunately, we've provided some facilities that can help you work it out. Your first at-a-glance debug feature is a group of three LEDs towards the butt end of the microcontroller module. When PixelPusher receives pixel data over the network, it lights these LEDs up (actually, to be specific, it toggles them each time it receives a packet). The three northernmost LEDs display in binary the number of the most recent strip update. The remaining LED is an indicator for the canned file playback mode.

If that doesn't help, there's always the big mighty sledgehammer: USB debugging. Take a mini USB cable and plug it into the USB client jack on the south end of the western edge of the device- the one that's near the microcontroller.

If you have a V2 board, you should see it pop up as a USB flash memory device. Be careful- this is where the firmware of the PixelPusher is stored. You can update the firmware by copying a file to this device, then dismounting it, "safely removing" it, or however your operating system calls it, and then pressing the reset button on the microcontroller. It's the one near the USB jack. PixelPusher always uses the newest file that ends in `.bin` for its firmware. You can also put your `pixel.rc` file on this flash, if you don't want to hook up a USB memory device, but we don't recommend it. If there's a `pixel.rc` file on both devices, the one on the USB stick is used in preference.

On the PixelPusher V3 board, we have a different scheme for updating the firmware, and you can store the `pixel.rc` file on a USB stick (just like the V2 board) or in the EEPROM on the Supervisor processor. This makes the boards much easier to handle in production, and allows us to make them \$20 cheaper. You will need to use a utility to write the configuration into the EEPROM.

Another thing that happens when you plug in the USB is that a USB-to-UART bridge becomes available. This looks a bit like a serial port, and you can connect to it with a serial terminal; the baud rate is 115200 and the framing is 8,n,1. When PixelPusher boots, it outputs a debugging console here; it can help you figure out why your configuration is not working. Each configuration decision taken during boot is listed, and the system's reason for rebooting is given too. If you're using Windows, you might need a .inf file to get the device to work; contact us for this file.

If you're contacting support because your device isn't behaving the way you expect, we'll ask you for the console log- just cut and paste it out of your terminal software and email it to us. We recommend that you use CoolTerm if you are using a Mac, or PuTTY if you are using Windows; if you're on Linux, then you can use screen or minicom.

