



Mini Thermal Receipt Printers

Created by Phillip Burgess



<https://learn.adafruit.com/mini-thermal-receipt-printer>

Last updated on 2022-12-16 08:45:17 PM EST

Table of Contents

Overview	5
• Models	
Power	7
• Making the Power Connection	
First Test	11
• Print Test Page	
Making Connections	14
• TTL Serial	
• For Product #597: “Mini”	
• For Product #2751: “Tiny”	
• For Product #2752: “Nano”	
• For Product #2753: “Guts”	
• To Arduino	
• To Raspberry Pi	
• To USB-to-Serial Cable (FTDI, etc.)	
• “Tiny” Printer: USB to Linux (Raspberry Pi, etc.)	
Arduino Code	23
• Install Library	
Printing Text	26
Bitmap Printing	27
• Windows	
• Mac and Linux	
Barcode Printing	31
CircuitPython and Python	32
• CircuitPython Installation of Thermal Printer Library	
• Python Installation of Thermal Printer Library	
• Firmware Version and Baud Rate	
• Usage	
• Create ThermalPrinter Class	
• Initialize ThermalPrinter	
• Test Page Print	
• Printer Fonts	
• Print Barcodes	
• Check Paper	
Python Docs	42
Downloads	42
• Firmware Changer	
Troubleshooting!	43

- [Parts and Tools Needed](#)
- [Procedure](#)
- [Code Changes](#)
- [Printing Huge Images](#)
- [Other Things to Know](#)

Overview



Add printing to any microcontroller project with these very cute thermal printers. Also known as receipt printers, they're what you see at the ATM or grocery store. Now you can embed a little printer of your own into a project. These printers is ideal for interfacing with a microcontroller, you simply need a 3.3V to 5V TTL serial output from your microcontroller to print text, barcodes, bitmap graphics, even a QR code!

These printers use very common 2.25" (58mm) wide thermal paper, available in the Adafruit shop or most office or stationery supply stores. You will also need a 5 Volt to 9 Volt regulated DC power supply that can provide 2 Amps or more during high-current printing — [our 5V 2A power supply will work very nicely \(http://adafru.it/276\)](http://adafru.it/276).

[You can pick up a thermal printer pack including printer, paper, power supply and terminal-block adapter in the Adafruit shop! \(http://adafru.it/600\)](http://adafru.it/600)

Of course, we wouldn't leave you with a datasheet and a "good luck!" — this tutorial and matching Arduino library demonstrate the following:

- Printing with small, medium and large text
- Bold, underline and inverted text
- Variable line spacing
- Left, center and right justification

- Barcodes in the following standard formats: UPC A, UPC E, EAN13, EAN8, CODE39, I25, CODEBAR, CODE93, CODE128, CODE11 and MSI - with adjustable barcode height
- Custom monochrome bitmap graphics
- How to print a QR code

Models



Our [Mini Thermal Receipt Printer](#) () is a popular choice as it accommodates a [full-length thermal roll \(15m/50'\)](#) (), meaning fewer paper changes. It's also available in a [starter pack](#) () that includes a 5V power supply and DC jack adapter.

This model has a 3-pin serial interface for connecting to 3.3V or 5V microcontrollers or Raspberry Pi.



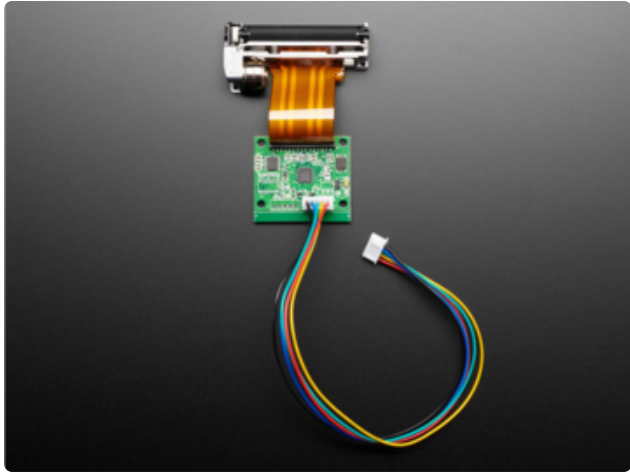
The [Tiny Thermal Receipt Printer](#) () is a bit more compact, accommodating shorter [10m/33' rolls](#) ().

What's interesting and unique to this model is that it has both a 5-pin serial header and a USB port, which can make for easier interfacing on Raspberry Pi.



For the most compact and portable situations, the [Nano Thermal Receipt Printer](#) () is smaller still, accommodating a [5m/16' paper roll](#) ().

This one has a 5-pin header for serial data and power.



For the most peculiar situations not covered above, the [Thermal Printer Receipt Guts \(\)](#) is just the insides of a thermal printer. You will need to design and build an enclosure to mount the hardware and hold a paper roll...in principle, any length roll can then work with this.

This unit has a 5-pin header for serial data and power.

The thermal paper rolls in the Adafruit shop are BPA-free and ready sized to each of the above units. You can also find compatible 2.25" (58mm) wide thermal paper at many office supply stores, though usually in 80–85 foot (25m) lengths that won't fit as-is in any of these printers. With some patience you can re-roll these onto an empty spool, cutting when the roll reaches a suitable diameter.

None of these small thermal printers have a cut feature; pull the finished print against the perforated edge. The Thermal Printer Guts has no perforated edge; you'll need to design this into your enclosure.

We offer documentation and guides for using these printers with ARDUINO, CIRCUITPYTHON and RASPBERRY PI. No support is provided for “native” OS printing in Windows, Android or macOS. Most can be used to some degree via the Blinka library for Python, with a USB-to-serial/FTDI cable or with the USB-equipped Tiny unit, but nothing beyond that scope is presented.

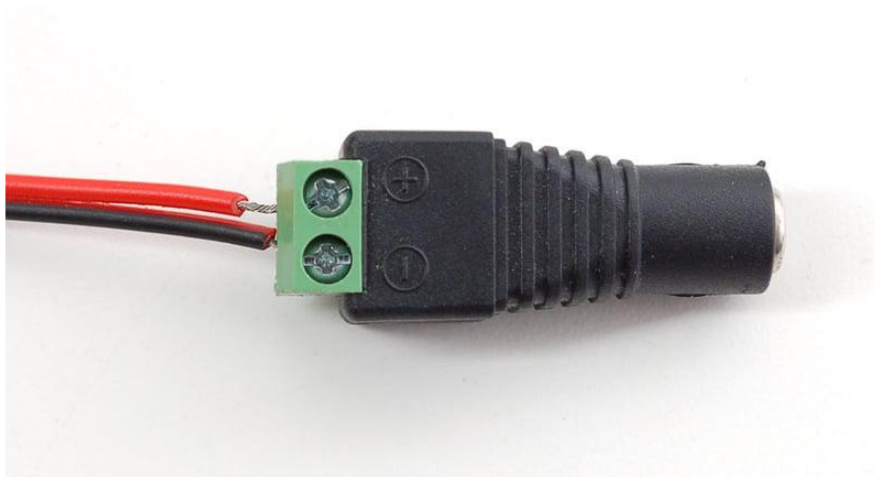
Power

These printers use a thermal head to heat the special receipt paper and draw images and text. That makes them very small — there's no moving ink head — but it means they require a lot of power: 5 to 9 Volts DC, at least 2 Amps of current. That means you will need a fairly beefy supply and you cannot run it off of USB power. An external adapter is required!



As a baseline, we suggest the [5V 2A power supply in our shop \(http://adafru.it/276\)](http://adafru.it/276). It provides sufficient power to keep the printer happy, and usually enough overhead to power a modest microcontroller project along with it.

A quick way to connect this to the printer's red/black power wires is with a [2.1mm jack adapter \(http://adafru.it/368\)](http://adafru.it/368):



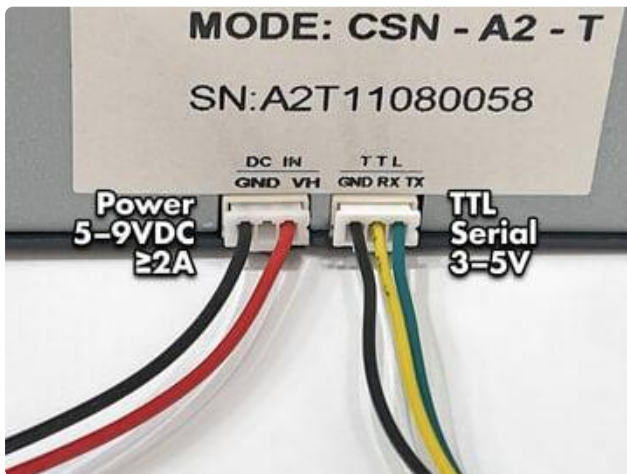
The 5V 2A supply mentioned above is a baseline. If using this, you might want to power the microcontroller separately through its USB port, to prevent a brownout condition when the printer is drawing a lot of current. Higher current supplies — [4 Amp \(\)](#) and [10 Amp \(\)](#)* — are also available in the shop, bulkier but with ample overhead for both the printer and supporting hardware you might be adding.

The higher current can make for darker printing and quicker, more assertive paper feed. Slightly higher voltage can too — to a point. One should never exceed 9 Volts to

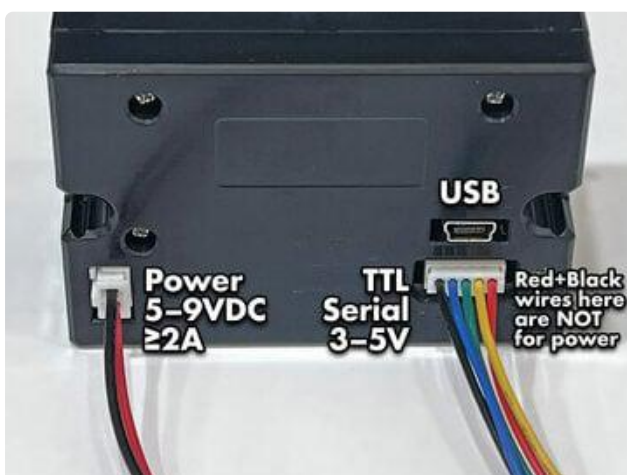
the printer; 12V or more will permanently damage it! 2A (or more) current is still recommended at these higher voltages. If upcycling a power supply from something else, carefully check the polarity (most, but not all, are “tip positive”) and that the output is DC, not AC.

A benefit to 5V power is that many dev boards can run from this directly and will regulate to 3.3V if needed; only a single power source is needed. That’s not necessarily true with higher voltages...though some, like the Arduino Uno, can accept 9V through the board’s DC power jack.

* The printer will never draw anywhere near this much current, but larger supplies are less susceptible to voltage droop when there’s a sudden demand. Sometimes just a matter of what’s presently in stock without waiting, or what other projects you might want to use this with in the future (e.g. large NeoPixel projects could benefit from the extra capacity of a 4A or 10A 5V supply).

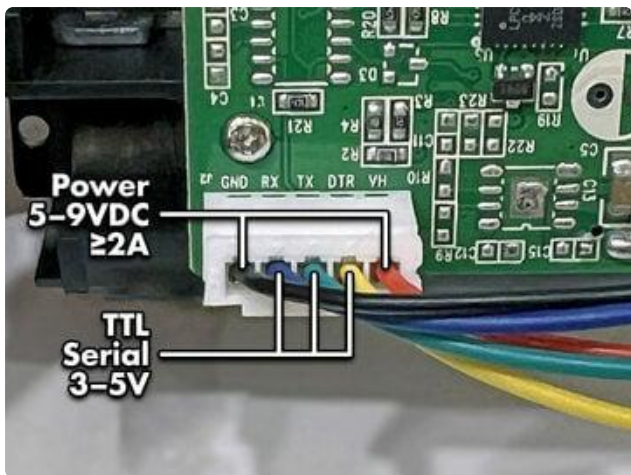


The Mini thermal printer has two 3-pin sockets—one for power, one for serial communication. Power goes to the jack labeled “DC IN.” Only the outer two pins of this connector are used.



The Tiny printer has three sockets: a 2-pin connector for power, 5-pin for TTL serial communication, and a USB port.

DO NOT attempt powering from the red and black wires of the serial cable. This cable is not following common electronics color conventions and connecting power here may damage the printer! Use ONLY the 2-pin connector for power.



The Nano printer and “printer guts” have a single 5-pin connector for power and serial data. The outermost red and black wires are for +V and ground, following typical conventions.

Making the Power Connection



During initial testing and for quick temporary projects, it’s usually okay to jam [breadboard prototyping wires](#) into the end connector on the DC power cable (or power+serial cable on Nano and guts), and the other end to a DC barrel jack adapter.

However, breadboard wires are small gauge and not suited to continuous heavy power draw. Once you’re “moved in” — confident the printer works and will suffice for a project — one option is to cut the DC cable, strip the wire ends and screw these directly to the DC barrel jack, without the breadboard wires as a go-between.

Keep the serial wires intact though. Breadboard wires work splendidly there.



The most robust option, if you want to get all fancy, is to splice a DC jack pigtail to the power cable.

This is not a part we carry. One option is to cut a [DC barrel jack extension cable](#) () in half and use the socket end. Or, the pigtails are commonly used in security camera installations, so you might find something by searching around.

First Test

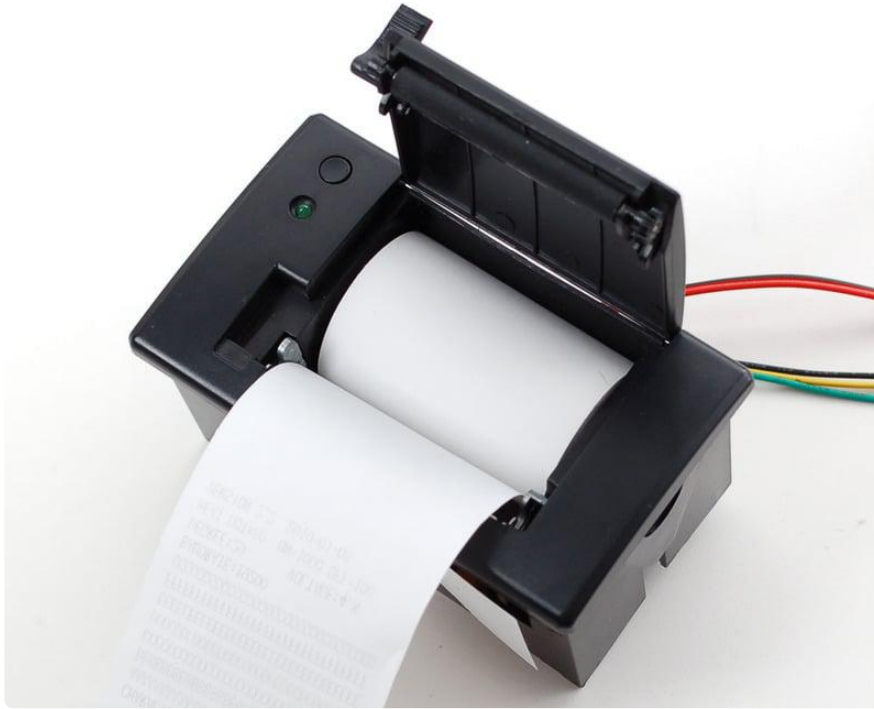
The first test you should do will ensure that paper is loaded and power is wired up correctly and the printer is running.

First up, pull the little plastic tab up top to open up the paper holder. Then insert a roll of 58 millimeter (2 1/4 inch) thermal paper into the bay as shown in the image below.

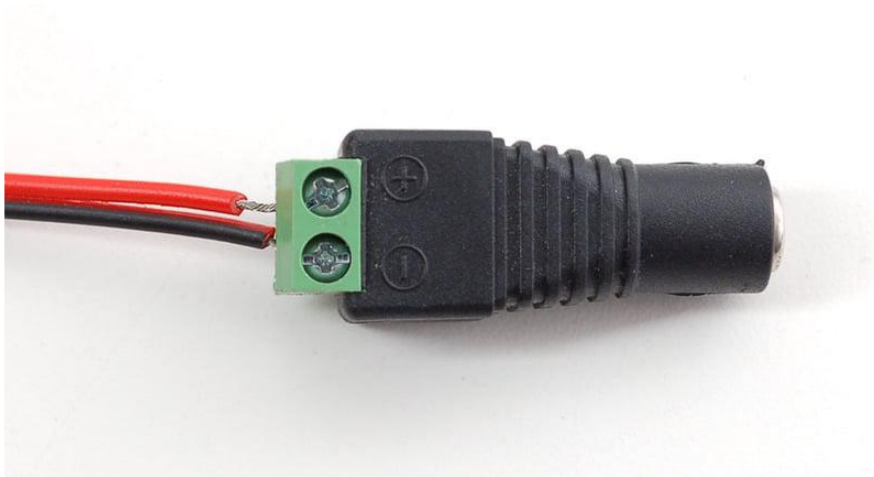
To reiterate from the first page, the printer models and corresponding roll sizes are:

- [Mini](#) (): [15m / 50 foot roll](#) ()
- [Tiny](#) (): [10m / 33'](#) ()
- [Nano](#) (): [5m / 15'](#) ()
- [Guts](#) (): no paper bay; use any size roll

Make sure that the paper doesn't bind or stick in the bay, it should rotate freely.



As described on prior page, power the printer using a 5V to 9V 2A or higher power supply, such as wiring up a 2.1mm DC power jack:



Print Test Page

Most of these thermal printers have a test feature providing basic diagnostics. You'll then know that power is good, paper is correctly loaded, and have some configuration details of the particular unit.

On most Mini and Tiny printers: hold down the button on the top of the printer while connecting power to print a test page. In normal use, tapping this button feeds the paper by one line.

On most Nano printers and printer guts: connect power first, then tap the button on the control board to print a test page.

Not all “printer guts” units have a button installed. If needed, you can make a temporary connection across the “J1” pads using a jumper wire or metal tweezers.

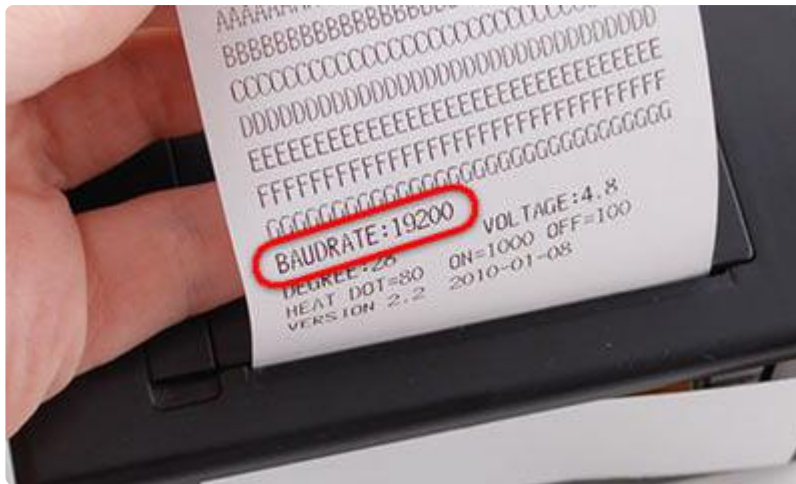
Occasional factory production changes occur...if you don't get a test page using the method above for you model, try the opposite method. A few may even lack a test page feature, but the status LED provides an indication of good power.



The status LED may blink when the printer is powered. It will not necessarily be on steady! This varies from model to model.

If you don't get a printout, check that the paper is inserted correctly and not binding, that the power is correctly wired, power supply is plugged in, etc. Then try again.

Note the baud rate on the test page. This may be 19200 or 9600. We'll need this number later:



If no baud rate is shown, or your printer has no self-test page feature, first assume 9600 as the default. In that doesn't work in tests that follow, then try 19200.

Making Connections

These printers might see use with microcontrollers or with Linux-based computers such as Raspberry Pi. The Tiny printer has the distinct option of connecting to a Raspberry Pi's USB port, covered later on this page. But to start, let's talk about TTL serial—the non-USB connection present across all of these printers.

TTL Serial

Most microcontrollers can provide a 3.3V or 5V TTL serial connection. It's not same as the 10V RS232 serial from a computer's 9-pin serial port — don't connect the printer directly to a standard PC port or you may damage it. A USB-to-serial cable (such as those made by FTDI, or similar) is an option on Linux PCs, but most will be using it with a microcontroller.

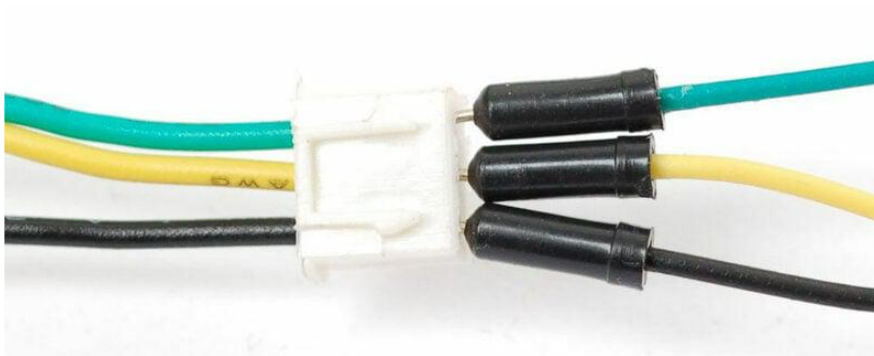
It makes no difference if the controller is a 5 Volt or 3.3 Volt device. The printer logic is 3.3V, but "5V tolerant," meaning no extra level-shifting circuitry is needed in either direction. Some older documentation may still show a voltage divider (two resistors and some extra wires), but that's since been found unnecessary. The printer and 5V or 3.3V devices can connect directly.

To start, we'll connect to the data cable of the printer. Easiest is to simply press breadboard jumper wires into the data plug, then use those to extend the connection to the host: an Arduino or other microcontroller, Raspberry Pi, or USB-to-serial cable.

If the controller is on a breadboard or the socket headers along the edges of an Arduino Uno-like board, or with most USB-to-serial cables: male/male jumper wires are suitable.

If the controller is a Raspberry Pi: use female/male jumper wires to the Pi's GPIO header.

The wire color for each pin is not the same across all these printers. Please read the descriptions carefully. It's nice (but not mandatory) to match jumper wire colors to the printer's data cable, to help keep track of things.



For Product #597: “Mini”

The Mini printer data cable has three wires:

- Black = GROUND reference
- Yellow = data IN to the printer (RX)
- Green = data OUT of the printer (TX)

This is in addition to the separate power cable, described on the prior page.

For Product #2751: “Tiny”

The Tiny printer data cable has five wires:

- Red = GROUND (yes, red, and yes this totally is the opposite of common electronics conventions)
- Green = data IN to the printer (RX)
- Blue = data OUT of the printer (TX)
- Yellow (DTR) and black can be left unconnected

This is in addition to the separate power cable, described on the prior page.

For Product #2752: “Nano”

The Nano printer cable has five wires and no separate power cable.

- Black = power and signal GROUND
- Blue = data IN to the printer (RX)
- Green = data OUT of the printer (TX)
- Red = 5–9V DC power
- Yellow = DTR, can be left unconnected for now

Power and data share a single connector here. Ground must go to both the power supply and the microcontroller, meaning you’ll need a split here, perhaps using a breadboard’s power rail.

For Product #2753: “Guts”

The Printer Guts cable has five wires and no separate power cable.

Some printers have a color-coded cable, while others have just have a plain white cable. Unplug this temporarily from the printer and you should find the pin functions labeled on the PCB:



The pin order here is not the same as the Tiny or Nano printers. And if you have both Guts and Nano printers, be super careful about the wiring, because the plugs are

interchangeable but the sequence of wires and colors (if any) do not have the same functions!

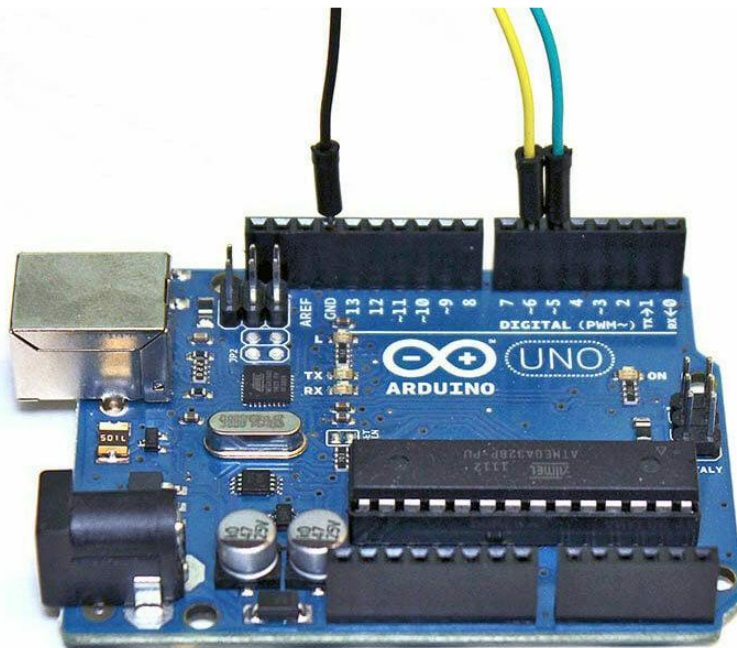
- VH = 5–9V DC power
- GND = power and signal GROUND
- RX on the PCB = data IN to the printer
- TX on the PCB = data OUT of the printer
- DTR on the PCB = can leave unconnected for now

As with the Nano printer, power and data share a single connector. Ground must go to both the power supply and the microcontroller, meaning you'll need a split here, perhaps using a breadboard's power rail.

To Arduino

For a board like the Arduino Uno, the other end of the jumper wires can insert into the board edge sockets. For smaller devices on a breadboard, insert into the corresponding contact strip.

Ground from the printer always connects to GND on the microcontroller board. For the data wires (TX and RX), which pins to connect to can vary by hardware and software, explained further on subsequent pages. Some situations may require specific pins, but you can usually use any two pins. Here's the printer's TX line connected to digital pin 5, and RX to digital pin 6:

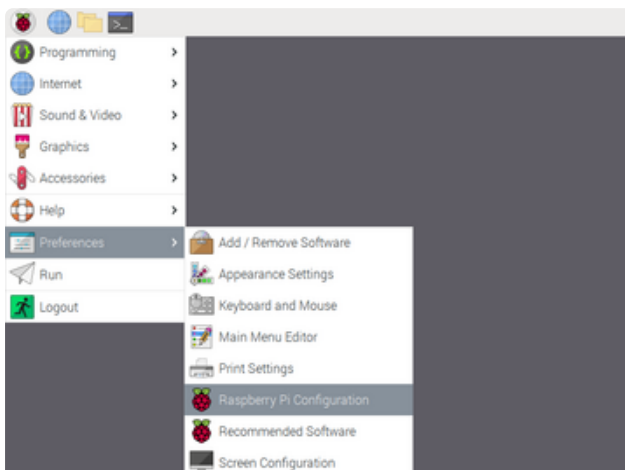


Note the colors of the wires above don't necessarily match the colors of every data cable, so double check the in and out designation for your product, they're all different!

To Raspberry Pi

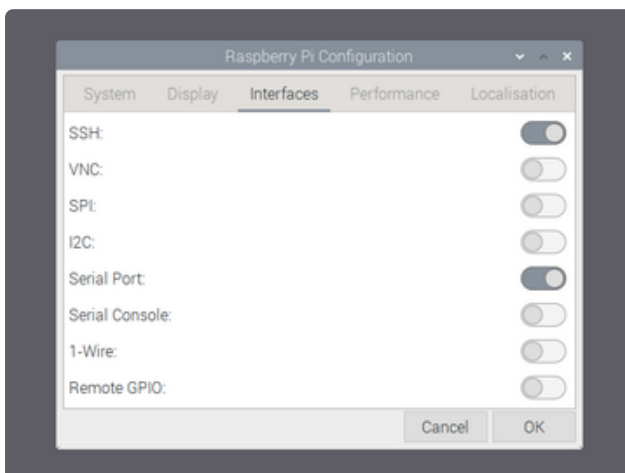
Before connecting any TTL serial printers to Raspberry Pi, it's vitally important to perform a little system configuration first.

The Raspberry Pi board has a TTL serial connection on the GPIO breakout header. By default, it's configured for a serial console — connected to a terminal, this provides another way to log into the system (along with Ethernet, WiFi or keyboard+monitor). We need to turn off the serial console behavior, or a connected printer will just spit out endless gibberish! This only applies to a TTL serial connection; Tiny printers using USB, or others when using a USB-to-serial adapter, don't need this step.



If using a desktop OS (mouse and GUI), it's just a few clicks. From the “Pi” menu at the top left, select Preferences→Raspberry Pi Configuration...

Select the Interfaces tab.



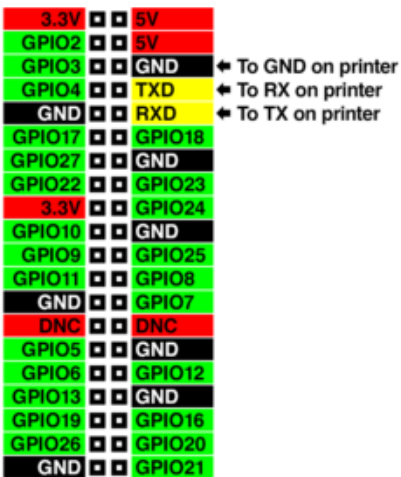
Turn Serial Port ON, and Serial Console OFF, as shown in the image.

Click the “OK” button, then “Yes” when asked whether to reboot

If a “lite” OS (text login), this is done from the command line:

```
sudo raspi-config
```

You'll find the Serial Port settings under "Interface Options." Select "No" for the login shell, and "Yes" if asked about the serial port hardware (this option might not show up on older Pi models, where it's always on). Tab over to "Finish" and reboot when asked.

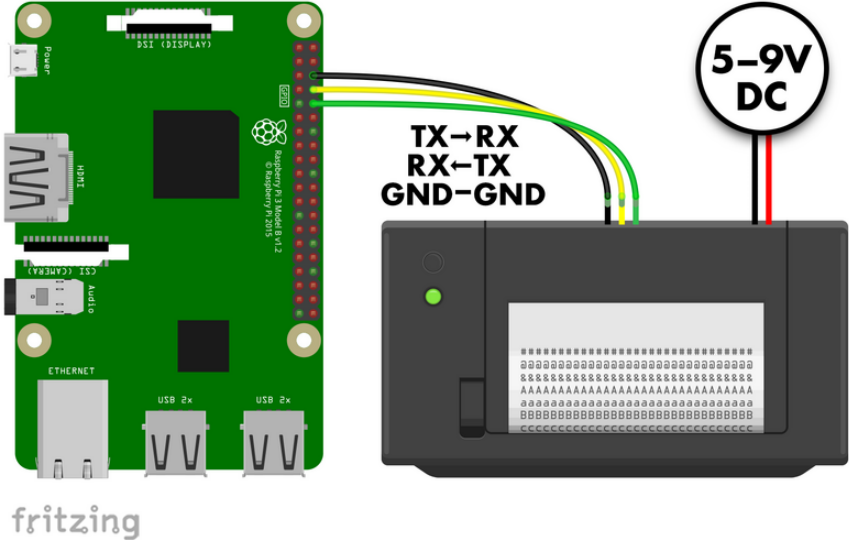


Now the printer can be connected without making a mess.

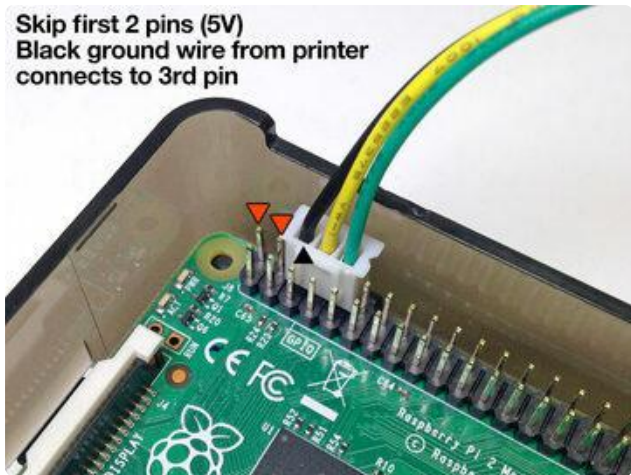
The diagram at left shows the Pi's GPIO header. For orientation reference, the 5V pin at the top right is nearest the corner of the board.

TX and TXD mean the same thing: transmit or transmit data. Likewise with RX and RXD. The terms might get used interchangeably.

TX and RX from the printer go to specific GPIO pins with opposite functions. TX to RX and RX to TX, known as a crossover configuration:



Really you can use any GND shown in the pinout diagram above. The third pin is easy to locate, but that spot's sometimes occupied by other hardware like small cooling fans.

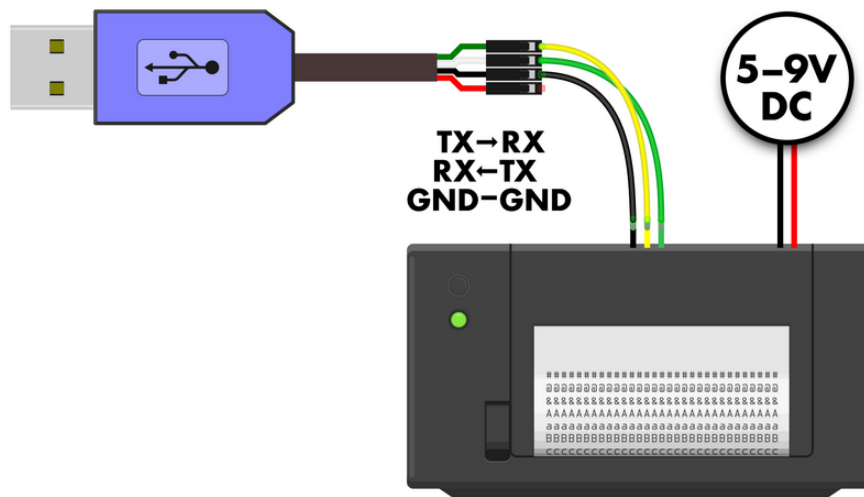


If you happen to be using the Mini printer with the 3-pin data cable, and if the Pi and printer are kept close together, you might not even need jumper wires. Skip the first two 5V pins on the GPIO header, and the data cable can press right on to the next 3 pins in sequence: GND, TX and RX.

To USB-to-Serial Cable (FTDI, etc.)

The principle here is similar to Raspberry Pi above: create a crossover by connecting the printer's TX to the adapter cable's RX pin, RX to TX, and ground to ground.

The diagram here is representative, not literal. Some adapter cables might have this pinout, but others will not. FTDI cables, for example, usually have a 6-pin header with its own color code; only 3 pins are needed here. You'll find pin functions and colors in the specific cable's documentation.



fritzing

Any of these cables will require device driver software. Popular ones (FTDI, Prolific, etc.) might already be present on Linux systems. Check documentation.

Also, how the device manifests when connected to a computer's USB port will vary among manufacturers and drivers. In Linux it might show as `/dev/tty.USBserial`

(followed by a number), but other conventions are sometimes used. Skim through the `/dev` directory and/or try the `lsusb` command to help identify the device.

“Tiny” Printer: USB to Linux (Raspberry Pi, etc.)

The Tiny printer has the lovely option of connecting via USB cable, no jumper wires needed. On Raspberry Pi, there's no need to change anything with raspi-config...but there are some things to know about the system.

On current Tiny printers, and on earlier ones when suitably configured, the printer appears to the system as `/dev/usb/lp0` and operates through the USB printing subsystem. Sometimes there will be a different number at the end, if more printers are attached.

On earlier Tiny printers, the factory configuration instead has it appear as `/dev/ttyUSB0`, which mimics a Prolific USB-to-serial bridge. Sometimes there will be a different number at the end, if other USB-to-serial devices are attached.

Any software or commands that want to issue data to the printer should speak to the appropriate device name (try `ls /dev` from the command line to see what's present, or try `lsusb`). Project code will often have the system device name in a global variable somewhere. Aside from that difference, they should both function the same; open the device, issue data, printer prints it.

If you have an older Tiny printer (defaulting to USB-to-serial bridge) and specifically require USB printing subsystem compatibility instead, there's a way to switch it over:

1. Similar to the self-test, hold the paper feed button while applying power...but now keep holding it.
2. After the QR code, the current USB mode is printed: this will either be COM (USB-to-serial mode, the default) or Printable Port (USB printing subsystem).
3. Release the paper feed button.
4. To keep the current setting, tap the paper feed button once more.
5. To change to to the opposite USB setting (COM or Printable Port), hold the feed button for at least 2 seconds. The printer will confirm the change.

FD437

KEEP HOLDING BUTTON

CHARACTER CODE TABLE

0-							
1-							
2-	!	"	#	\$	%	&	'
3-	0	1	2	3	4	5	6
4-	@	A	C	D	E	F	G
5-	P	Q	S	T	U	V	W
6-	`	a	b	c	d	e	f
7-	p	q	r	s	t	u	v
8-	Ç	ü	ä	å	ç	è	é
9-	É	ê	ë	ö	ù	ý	ÿ
A-	á	í	ñ	ñ	ñ	ñ	ñ
B-	ü	ü	ü	ü	ü	ü	ü
C-	ü	ü	ü	ü	ü	ü	ü
D-	ü	ü	ü	ü	ü	ü	ü
E-	α	β	γ	δ	ε	ζ	η
F-	±	±	±	±	±	±	±

FontB:

!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJ
 KLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

Baudrate:9600

DEGREE:34 VOLTAGE:4.9

HEAT DOT=8 ON=110 OFF=100

VERSION:1.11 2013-11-13



UNTIL YOU GET:

Current USB:Printable Port

Press FEED 2s to Shift USB Func

Press FEED less 2s to exit

HOLD AGAIN FOR 2 SECS

Current USB:COM

****Exit****

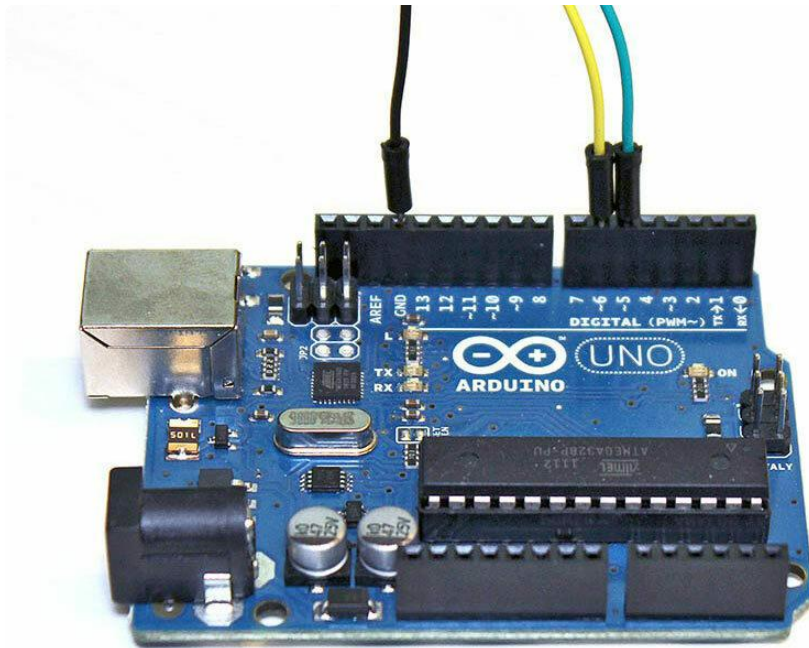
DONE!

This is present only in older Tiny printers. Current units work strictly in USB printing subsystem mode—it's considered more modern—and continuing to hold the paper feed button will just eject a lot of paper...so don't.

Arduino Code

Connections are explained on the prior page. For the Arduino example code, we'll connect the printer's data out (TX) wire to digital pin 5, and data in (RX) to digital pin 6. Ground can go to any Arduino GND pin. Pins are configurable, but this is what the example uses, so let's stick with that for now.

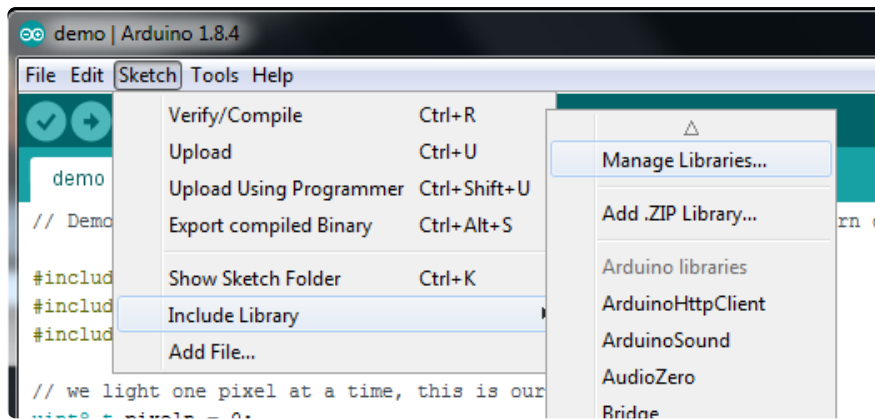
On the Arduino side, pin 5 receives (RX) and 6 transmits (TX)...opposite of the printer's pin functions...called a crossover configuration, something you'll hear a lot in serial communication and networking.



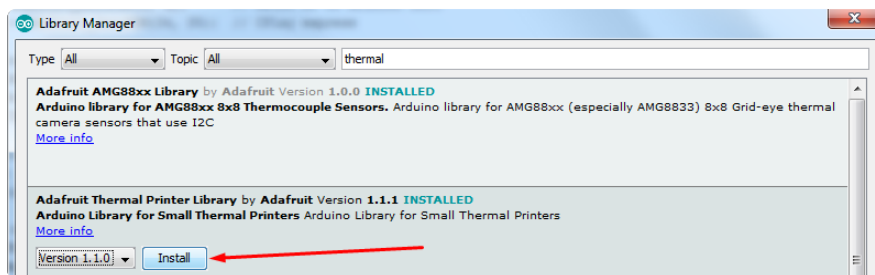
Install Library

We made an [Arduino library to assist working with these printers \(\)](#).

Go to the Arduino Library Manager under Sketch→Include...Library→Manage... Libraries...



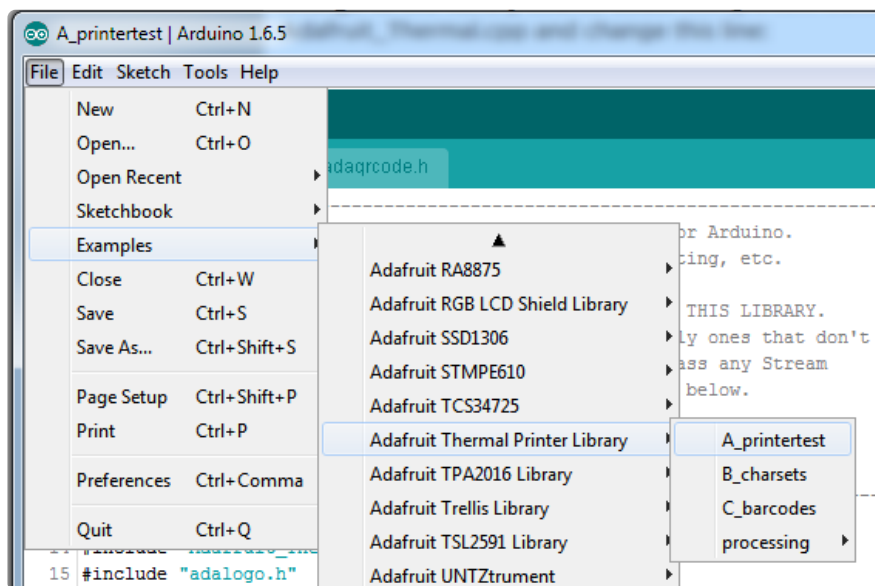
Search for and install the Adafruit Thermal Printer library:



We also have a great tutorial on Arduino library installation at:

<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use> ()

You should now be able to access the sample code by navigating through menus in this order: File→Sketchbook→Libraries→Adafruit_Thermal→A_printertest



If your printer test page shows 'BAUDRATE: 9600', you'll need to make a small change to the library source code. Using a text editor (Notepad, etc.) open the file Adafruit_Thermal.cpp and change this line:


```
#define BAUDRATE 19200
```

to this:

```
#define BAUDRATE 9600
```

Some printers arrive from the factory set for 19200 baud, but a few may be set to 9600. This will not negatively impact the performance of your unit! The speed of the paper through the printer is already much less than this and you will not see any difference...it's strictly a data protocol issue of getting the microcontroller and printer communicating.

OK upload the sketch (with baudrate change if necessary) to the Arduino. You should see the printer print out the example receipt which includes all the capabilities of the library.



If this does not work, first check that the printer and Arduino are both powered, and that the data in, out and ground wires are properly connected to the Arduino.

Printing Text



The thermal printer has a few handy things it can do, most of which are in the `A_printertest` sketch. These are shown in the image above. In order, starting from the top:

- Inverted text: this is invoked by calling `inverseOn()` — you will get text that’s white-on-black instead of black-on-white. `inverseOff()` turns this off.
- Double height: this makes text that’s extra tall, call `doubleHeightOn()` — likewise, turn off with `doubleHeightOff()`
- Left/Center/Right justified: this aligns text to the left or right edge of the page, or centered. You can set the alignment by calling `justify('R')` (for right-justified), `justify('C')` (for centered) or `justify('L')` (for left-justified). Left-justified is the default state.
- Bold text: makes it stand out a bit more, enable with `boldOn()` and turn off with `boldOff()`
- Underlined text: makes it stand out a bit more, enable with `underlineOn()` and turn off with `underlineOff()`
- Large/Medium/Small text: by default we use small, medium is twice as tall, large is twice as wide/tall. Set the size with `setSize('L')`, `setSize('M')` or `setSize('S')`
- Line spacing: you can change the space between lines of text by calling `setLineHeight()` where `numpix` is the number of pixels. The minimum is 24 (no extra space between lines), the default spacing is 32, and double-spaced text would be 64.

Look through the source of the `A_printertest` sketch to see these used in context.

Bitmap Printing

This printer can produce bitmaps, which can add a touch of class to a receipt with your logo or similar.

The first step is to get the image prepared. The printer can only do monochrome (1-bit) images, and the maximum width is 384 pixels. We suggest starting with a small bitmap (100 pixels or less on each side) and then experimenting to get the size and look you want.

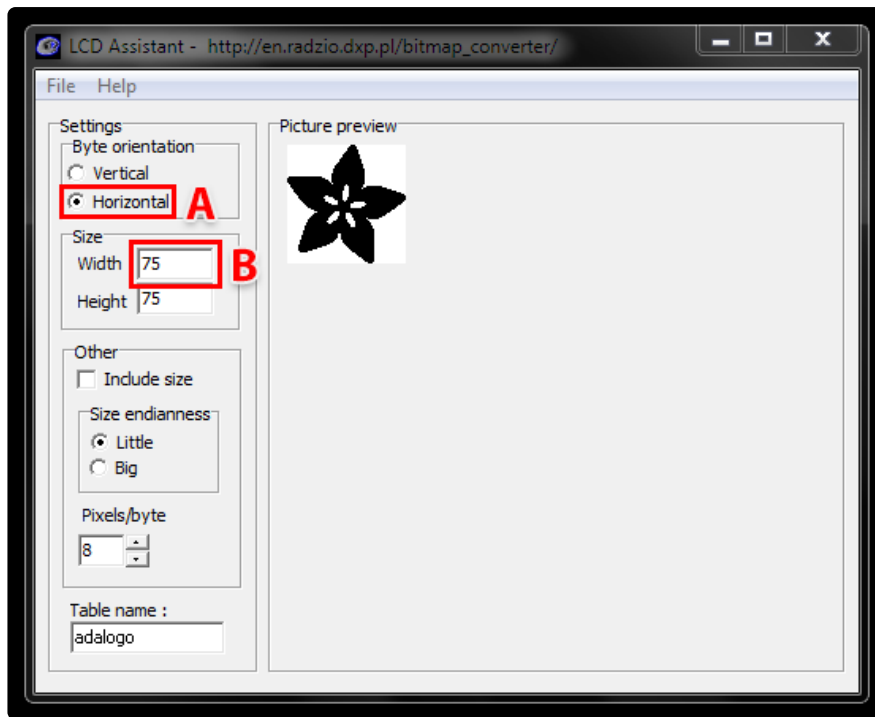
A few steps are required to prepare an image for printing. For Windows users, there's a nice graphical user interface for this. For Mac and Linux, different tools are used... not as visually slick, but they do the job well.

These barcode printers shouldn't be confused with a LaserJet - they're not good at printing heavy/dense images with lots of black or they might stick and stall!

Windows

Use an image editing program to save your image as a 1-bit BMP — in Windows, the built-in Paint program will suffice.

Download, install and run [LCD Assistant](#) (). This program is for Windows only but does a really fantastic job! Load the BMP file you previously generated (in Paint, etc.). The file must be in BMP format — the software won't read PNG, GIF, etc. Then a couple of settings need to be adjusted...



First, in the “Byte orientation” section of the settings, select “Horizontal” (item A in the image above).

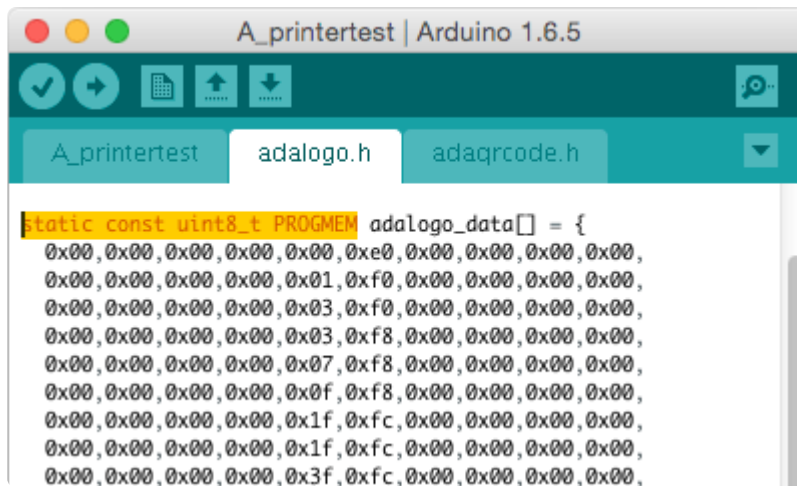
Second (item B above), you may need to change the Width setting. Because this software (and the thermal printer) handle images in horizontal groups of eight pixels, if the image width is not a multiple of 8, it will be truncated (cropped) to the nearest smaller 8-pixel boundary. For example, with the 75 pixel wide image above, the output will be cropped to only 72 pixels wide, losing some data from the right edge. To avoid this, increase this number to the next multiple of 8 (that would be 80 for the example above), and the output will be padded with blank pixels to cover the gap. Remember the number you use here, you’ll need it later.

The image height does not need to be adjusted this way, only width.

Set the table name to something short but descriptive (e.g. “adalogo” above), then select Save Output from the File menu. Give the file a similarly brief but descriptive name, ending in “.h” (e.g. “adalogo.h”).

To get this file into your Arduino sketch, select “Add File...” from the Sketch menu. This will add a new tab to your code. Your original code is still there under the leftmost tab.

A couple of small changes are now needed in both tabs. First, at the top of the file containing the new table data, change “const unsigned char” to “static const uint8_t PROGMEM” as shown below:



```
static const uint8_t PROGMEM adalogo_data[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x01, 0xf0, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x03, 0xf0, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x03, 0xf8, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x07, 0xf8, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x0f, 0xf8, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x1f, 0xfc, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x1f, 0xfc, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x3f, 0xfc, 0x00, 0x00, 0x00, 0x00,
```

Next, in the tab containing the main body of your code, add an “include” statement to reference the new file:

```
#include "adalogo.h"
```

Check the A_printertest example sketch if you’re not sure how to include the code properly.

You can now output the image by calling `printBitmap(width, height, tablename)`, where width and height are the dimensions of the image in pixels (if you changed the image width to a multiple of 8 as previously described, use that number, not the original image size), and tablename is the name of the array in the new tab (e.g. “adalogo” above).



Having a graphical user interface is nice, but some of these extra steps can be confusing and error-prone. If you prefer, the technique below for Mac and Linux works in Windows as well.

Mac and Linux

The conversion tool for Mac and Linux doesn't include a fancy GUI, but it works well and avoids several steps (and potential mis-steps). The source image doesn't need to be in BMP format — most image formats can be read natively — and the output can be added to a sketch with no further editing. It works for Windows as well, if you'd rather use this method.

First, if you don't already have the Processing language installed, [download it from processing.org](https://processing.org/) (). Processing looks almost exactly like the Arduino IDE, but it's for writing code for your normal computer, not a microcontroller. This can be a little confusing to first-timers, so if something doesn't seem to compile, make sure you're running code in the right environment: Arduino for the Arduino board, Processing for your computer.

This code runs in Processing 2.x, the latest version of Processing available from their website. Don't use the older Processing 1.5.x version!

The Adafruit_Thermal library folder that you previously downloaded contains a sub-folder called processing. Inside that is a sketch called bitmapImageConvert.pde. Load this into Processing and press RUN (the triangle button).

You'll be prompted to select an image using the system's standard file selection dialog. The program runs for just a brief instant, and will create a new file alongside the original image file. For example, if you selected an image called "adalogo.png", there will be a new file called "adalogo.h" in the same location. This file contains code to add to your Arduino sketch. You shouldn't need to edit this file unless you want to change the variable names within.

To get this file into your Arduino sketch, select "Add File..." from the Sketch menu. This will add a new tab to your code. Your original code is still there under the leftmost tab.

Next, in the tab containing the main body of your code, add an "include" statement to reference the new file:

```
#include "adalogo.h"
```

Check the A_printertest example sketch if you're not sure how to include the code properly.

If the source image was called `adalogo.png`, then the resulting `.h` file (`adalogo.h`) will contain three values called `adalogo_width`, `adalogo_height` and `adalogo_data`, which can be passed directly and in-order to the `printBitmap()` function, like this:

```
printBitmap(adalogo_width, adalogo_height, adalogo_data);
```

Barcode Printing

Thermal printers are really good at printing barcodes! This printer supports 11 different codes - UPC A, UPC E, EAN13, EAN8, CODE39, I25, CODEBAR, CODE93, CODE128, CODE11 and MSI. It only supports linear (1-D) barcodes, and can't generate 2-D barcodes like QR codes (although there is a hack you can do, see below!) Barcodes are generated "on the fly," which is nice — you can customize the height and data included quite easily.

You can make a barcode by calling `printBarcode("barcodedata", BARCODETYPE)`, where the first string is the data to encode (e.g. a UPC code) and `BARCODETYPE` can be `UPC_A`, `UPC_E`, `EAN13`, `EAN8`, `CODE39`, `I25`, `CODEBAR`, `CODE93`, `CODE128`, `CODE11` or `MSI`.

Some barcodes are very restricted — you can only put in 12 numbers, no characters. Others are very flexible and take nearly any character input. [Please check out the wikipedia list detailing kinds of barcodes](#) () to pick the right one for your application.

The available range of barcodes varies with the printer firmware revision. Check `Adafruit_Thermal.h` for a list of codes.

It's also possible to print QR codes, if you're willing to pre-generate them. This might be handy if you want to, let's say, include a URL on the receipt and the URL doesn't change. [You can generate QR codes at many sites including this one.](#) () Use the smallest QR code size. The image will be in PNG format, so if you're using the Windows LCD Assistant tool you'll need to convert it to BMP first (Windows Paint works for this). Then you can convert and embed this in your Arduino sketch as previously described.



CircuitPython and Python

It's easy to use the thermal printer with CircuitPython (or Python [via the Blinka library](#) ()) and the [Adafruit CircuitPython Thermal Printer](#) () module. This module allows you to easily write Python code that controls the printer, like printing text and barcodes (note right now bitmap printing is not supported by CircuitPython code).

Wiring is explained on the “Making Connections” page.

On Raspberry Pi, a common choice is to use the board's GPIO pins to connect to the printer's TTL serial port. Also on Pi, and on most other computer (not microcontroller) situations, there's the USB port on the Tiny printer, or using a USB-to-serial or FTDI cable. [Please visit the guide for CircuitPython on Linux to see whether your platform is supported](#) ().

CircuitPython Installation of Thermal Printer Library

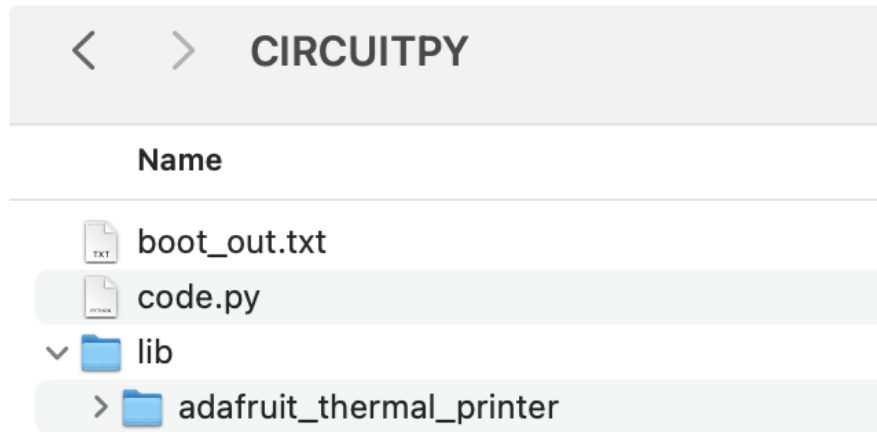
If you're using a CircuitPython microcontroller, you'll need to install the [Adafruit CircuitPython Thermal Printer](#) () library on your CircuitPython board.

First make sure you are running the [latest version of Adafruit CircuitPython](#) () for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle](#) (). Our introduction guide has [a great page on how to install the library bundle](#) () for both express and non-express boards.

You can also download the `adafruit_thermal_printer` folder from [its releases page on Github \(\)](#).

Copy the `adafruit_thermal_printer` folder into the CircuitPython drive's `lib` directory. Place the whole folder itself there, not the individual files within.



Python Installation of Thermal Printer Library

If you're using a Raspberry Pi or other computer with Python, you'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may require verifying you are running Python 3. [Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready \(\)!](#)

Once that's done, from your command line run the following command:

```
sudo pip3 install adafruit-circuitpython-thermal-printer
```

Firmware Version and Baud Rate

Before you can use the printer module you must determine the firmware version and baud rate of the printer. This is easy to find out by printing a test page from the printer as described on the ["First Test" page \(\)](#), hold the printer button as power is applied. Print out the test page and take note of these values printed at the bottom:

- Baudrate - Remember this value, typically 19200 or 9600.
- Version - Take note of this value too, it's a value like 2.2, 2.64, 2.68, etc. You will use the version to pick the appropriate module for interacting with the printer.



Usage

If you're using a CircuitPython microcontroller, [connect to the board's serial REPL](#) (so you are at the CircuitPython >>> prompt. If you're using a computer like the Raspberry Pi, bring up a REPL at the command prompt by typing `python3`. (On some systems, `python` may run Python 3 - just make sure you're not using Python 2, as it isn't supported.)

To demonstrate the usage of the sensor we'll initialize it and print some text from the REPL.

First you must import the necessary modules to initialize the hardware serial connection to the printer. These commands will vary depending on your hardware - choose the appropriate set:

```
# On a CircuitPython microcontroller:
import board
import busio
uart = busio.UART(board.TX, board.RX, baudrate=19200)
```

```
# On the Raspberry Pi with built-in UART:
import serial
uart = serial.Serial("/dev/serial0", baudrate=19200, timeout=3000)
```

```
# On the Raspberry Pi with the USB-to-serial converter:
import serial
uart = serial.Serial("/dev/ttyUSB0", baudrate=19200, timeout=3000)
```

```
# On the Raspberry Pi with recent Tiny printers on USB:
import serial
uart = serial.Serial("/dev/usb/lp0", baudrate=19200, timeout=3000)
```

Notice the baud rate of the serial connection is specified with the `baudrate` keyword. For most printers they'll use a rate of 19200 baud, however other printers might be configured to use a different rate like 9600 baud. Use the baud rate you saw printed on the test page of your printer!

Also be aware CircuitPython currently requires bi-direction UART connections so even if you don't have the board RX pin hooked up (as recommended) you must keep the RX pin disconnected and ignore using it in your project.

Create ThermalPrinter Class

Next you'll need to import the thermal printer module and call a function to get the right class depending on the version of your printer firmware. Be sure you've found the exact version number for your printer by following the steps mentioned above and on the previous test page!

```
import adafruit_thermal_printer
ThermalPrinter = adafruit_thermal_printer.get_printer_class(2.69)
```

Notice you call the `get_printer_class` function from within the `adafruit_thermal_printer` module. You must pass this function the version of the thermal printer that you're using and it will internally find the right class to import and use in your code. In this example we're specifying a printer with version 2.69 firmware, but be sure to specify your printer's version!

The result is a class which you can immediately initialize:

Initialize ThermalPrinter

Now construct an instance of the `ThermalPrinter` class:

```
printer = ThermalPrinter(uart)
```

Notice you must pass in the serial UART connection that was previously constructed.

Be aware it will take from a half to even 5 seconds for the printer to be created and initialize itself and warm up. You can optionally specify `auto_warm_up=False` as a

keyword and manually call the `warm_up` function to control when this warm-up time occurs if needed in your application (or you are confused why your code takes so long to start up).

Test Page Print

Now you're ready to print!

Try printing a full test page with the `test_page` function:

```
printer.test_page()
```

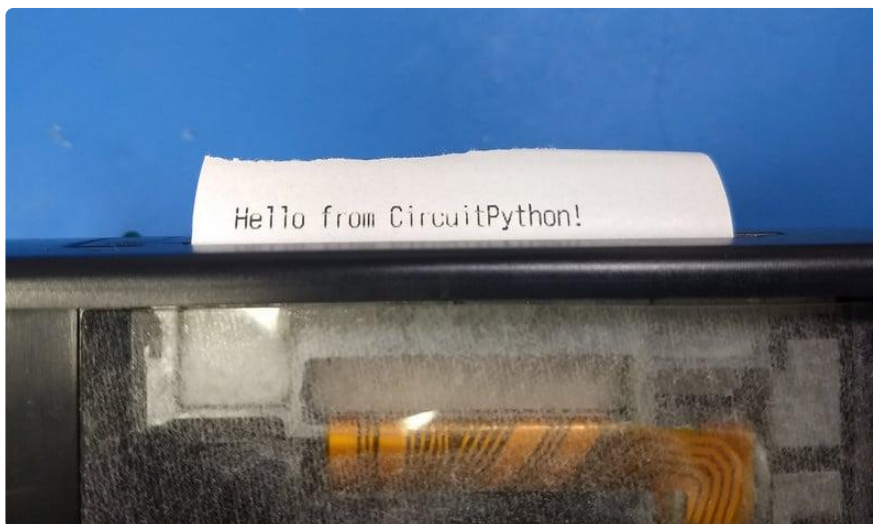
You should see the printer test page print out.

You can advance the paper forward a number of lines with the `feed` function, try feeding 2 lines to make some space:

```
printer.feed(2)
```

Now print a small line of text with the `print` function and feed a few lines to move it out of the printer to read:

```
printer.print('Hello from CircuitPython!')  
printer.feed(2)
```



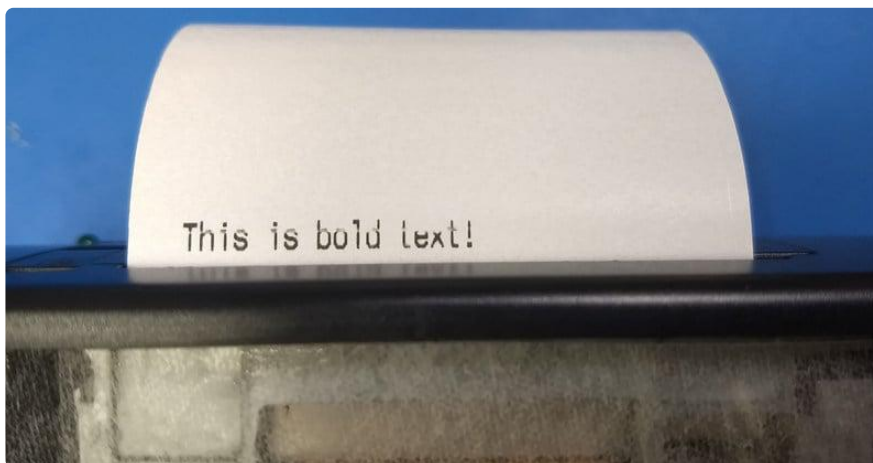
Printer Fonts

There are a few properties you can use to adjust how text is printed. Each of these can be set to a `True` value to enable, or a `False` value to disable:

- `bold` - Print bold text.
- `inverse` - Print inverted (white on black) text.
- `upside_down` - Print upside down text (might not work on all versions of printers).
- `double_height` - Double height size text.
- `double_width` - Double width size text.
- `strike` - Strike-through text (again might not work on all versions of printers).

Try turning on bold text and printing a line:

```
printer.bold = True # Turn on bold
printer.print('This is bold text!')
printer.bold = False # Turn off bold
# Feed lines to make visible:
printer.feed(2)
```



There are a few properties you can set to special values to further control text printing:

- `underline` - This controls underline printing and can be `None` (off), `adafruit_thermal_printer.UNDERLINE_THIN`, or `adafruit_thermal_printer.UNDERLINE_THICK`.
- `size` - This controls the size of text and can be `adafruit_thermal_printer.SIZE_SMALL`, `adafruit_thermal_printer.SIZE_MEDIUM`, or `adafruit_thermal_printer.SIZE_LARGE`. The default is small.

- `justify` - This controls the justification or location of printed text and can be `adafruit_thermal_printer.JUSTIFY_LEFT`, `adafruit_thermal_printer.JUSTIFY_CENTER`, or `adafruit_thermal_printer.JUSTIFY_RIGHT`. The default is justify left.

For example to print thick underlined, medium text, with center justification:

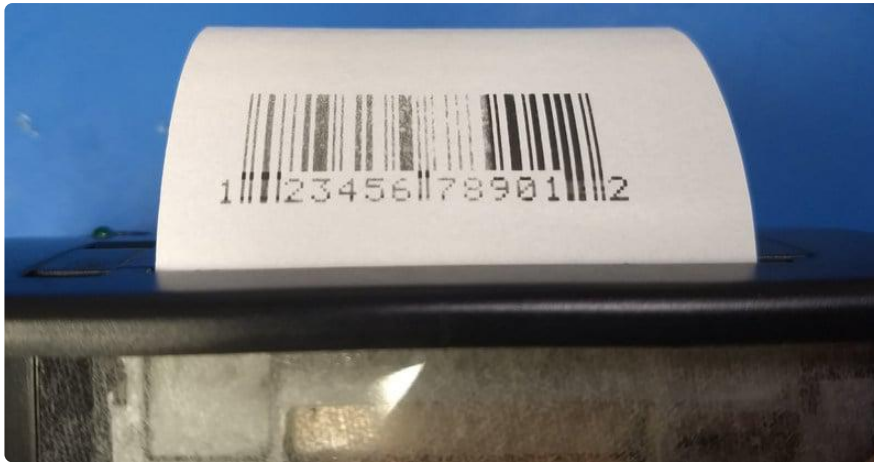
```
printer.underline = adafruit_thermal_printer.UNDERLINE_THICK
printer.size = adafruit_thermal_printer.SIZE_MEDIUM
printer.justify = adafruit_thermal_printer.JUSTIFY_CENTER
printer.print('Medium center!')
# Reset back to normal printing:
printer.underline = None
printer.size = adafruit_thermal_printer.SIZE_SMALL
printer.justify = adafruit_thermal_printer.JUSTIFY_LEFT
# Feed lines to make visible:
printer.feed(2)
```



Print Barcodes

Finally you can print a barcode with the `print_barcode` function. Each printer supports a different type and format of barcode so consult the product sheet in the downloads section for each type. However here's an example of printing a UPC A barcode with value '123456789012':

```
printer.print_barcode('123456789012', printer.UPC_A)
printer.feed(2)
```



The first parameter to `print_barcode` is a string that defines the barcode value (this varies depending on the type of barcode, a UPC A code is 13 digits). The second parameter is the type of the barcode, which again is dependent on the printer firmware. Typical values you might use include:

- `printer.UPC_A`
- `printer.UPC_E`
- `printer.EAN13`
- `printer.EAN8`
- `printer.CODE39`
- `printer.ITF`
- `printer.CODABAR`
- `printer.CODE93`
- `printer.CODE128`

Check Paper

With some printers (not all, and it varies among firmware releases), if you've connected the RX pin of your board to the printer TX, you can query paper status with the `has_paper` function that returns `True` or `False` if there is paper present:

```
printer.has_paper()
```

```
>>> printer.has_paper()
True
>>> █
```

That's all there is to using the thermal printer with CircuitPython!

A complete demo of the usage is in the [thermal_printer_simpletest.py demo in the library \(\)](#).

If you're running thermal_printer_simpletest.py on the Raspberry Pi (or any computer), you'll have to make some changes.

On the Raspberry Pi, comment out the `uart = busio(...)` line, and uncomment the `import serial` and `uart = serial.Serial(...)` lines, changing `/dev/ttyUSB0` to the appropriate serial port.

Unless you used a voltage divider to hook up the printer TX line, make sure you comment out the paper check code:

```
# Check if the printer has paper. This only works if the RX line is connected
# on your board (but BE CAREFUL as mentioned above this RX line is 5V!)
# if printer.has_paper():
#     print('Printer has paper!')
# else:
#     print('Printer might be out of paper, or RX is disconnected!')
```

Now you can run the program with the following command:

```
python3 thermal_printer_simpletest.py
```

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple demo of printer functionality.
# Author: Tony DiCola
import board
import busio

import adafruit_thermal_printer

# Pick which version thermal printer class to use depending on the version of
# your printer. Hold the button on the printer as it's powered on and it will
# print a test page that displays the firmware version, like 2.64, 2.68, etc.
# Use this version in the get_printer_class function below.
ThermalPrinter = adafruit_thermal_printer.get_printer_class(2.69)

# Define RX and TX pins for the board's serial port connected to the printer.
# Only the TX pin needs to be configured, and note to take care NOT to connect
# the RX pin if your board doesn't support 5V inputs. If RX is left unconnected
# the only loss in functionality is checking if the printer has paper--all other
# functions of the printer will work.
RX = board.RX
TX = board.TX

# Create a serial connection for the printer. You must use the same baud rate
# as your printer is configured (print a test page by holding the button
# during power-up and it will show the baud rate). Most printers use 19200.
uart = busio.UART(TX, RX, baudrate=19200)

# For a computer, use the pyserial library for uart access.
# import serial
```



```

# uart = serial.Serial("/dev/serial0", baudrate=19200, timeout=3000)

# Create the printer instance.
printer = ThermalPrinter(uart, auto_warm_up=False)

# Initialize the printer. Note this will take a few seconds for the printer
# to warm up and be ready to accept commands (hence calling it explicitly vs.
# automatically in the initializer with the default auto_warm_up=True).
printer.warm_up()

# Check if the printer has paper. This only works if the RX line is connected
# on your board (but BE CAREFUL as mentioned above this RX line is 5V!)
if printer.has_paper():
    print("Printer has paper!")
else:
    print("Printer might be out of paper, or RX is disconnected!")

# Print a test page:
printer.test_page()

# Move the paper forward two lines:
printer.feed(2)

# Print a line of text:
printer.print("Hello world!")

# Print a bold line of text:
printer.bold = True
printer.print("Bold hello world!")
printer.bold = False

# Print a normal/thin underline line of text:
printer.underline = adafruit_thermal_printer.UNDERLINE_THIN
printer.print("Thin underline!")

# Print a thick underline line of text:
printer.underline = adafruit_thermal_printer.UNDERLINE_THICK
printer.print("Thick underline!")

# Disable underlines.
printer.underline = None

# Print an inverted line.
printer.inverse = True
printer.print("Inverse hello world!")
printer.inverse = False

# Print an upside down line.
printer.upside_down = True
printer.print("Upside down hello!")
printer.upside_down = False

# Print a double height line.
printer.double_height = True
printer.print("Double height!")
printer.double_height = False

# Print a double width line.
printer.double_width = True
printer.print("Double width!")
printer.double_width = False

# Print a strike-through line.
printer.strike = True
printer.print("Strike-through hello!")
printer.strike = False

# Print medium size text.
printer.size = adafruit_thermal_printer.SIZE_MEDIUM

```

```
printer.print("Medium size text!")

# Print large size text.
printer.size = adafruit_thermal_printer.SIZE_LARGE
printer.print("Large size text!")

# Back to normal / small size text.
printer.size = adafruit_thermal_printer.SIZE_SMALL

# Print center justified text.
printer.justify = adafruit_thermal_printer.JUSTIFY_CENTER
printer.print("Center justified!")

# Print right justified text.
printer.justify = adafruit_thermal_printer.JUSTIFY_RIGHT
printer.print("Right justified!")

# Back to left justified / normal text.
printer.justify = adafruit_thermal_printer.JUSTIFY_LEFT

# Print a UPC barcode.
printer.print("UPCA barcode:")
printer.print_barcode("123456789012", printer.UPC_A)

# Feed a few lines to see everything.
printer.feed(2)
```

Python Docs

[Python Docs \(\)](#)

Downloads

- [Adafruit_Thermal library for Arduino \(\)](#).
- [LCD Assistant \(\)](#) — optional bitmap conversion utility for Windows.
- [Processing \(\)](#) language — needed for bitmap conversion for Mac or Linux (and optionally Windows). DOWNLOAD VERSION 1.5.1, not the 2.0 beta.
- [Thermal Printer User Manual \(\)](#).
- [An older version of the Thermal Printer User Manual \(\)](#).
- [Thermal Printer Product Sheet \(\)](#).

Firmware Changer

If you'd like to risk it, you can revert/change the firmware using this tool that is from the manufacturer. We haven't tried it. It's completely utterly unsupported and you may break your printer. We won't replace it if it stops working due to a firmware change!

[update_firmware.zip](#)

Troubleshooting!

I'm not getting it to work!

First thing to try is the power up test. You do not need an Arduino for this.

Make sure the paper is in the bay correctly, it should feed under and up so that the paper comes through the slot without bending.

For mini and tiny printers, hold down the button on the top, then plug in the printer to power. It should print out a test page. On nano printers and guts, connect power first, then tap the button.

The printer's LED will not be on solid, it will blink once in a while, that's normal.

I'm trying the self test and it's still not working.

Make sure you have a good power supply that can deliver the required current. If you are trying to operate at 5V, your power supply must be able to maintain a constant 5V even when the current demand spikes during printing. If your printer is making an attempt to print, but then struggling, then it is likely your power supply is dipping below 5V.

If you have one available, try a higher voltage power source, but no more than 9 Volts.

I'm having difficulty getting a printout.

Make sure the paper is in the bay correctly, it should feed under and up so that the paper comes through the slot without bending.

Make sure the paper roll is not 'stuck' feeling in the bay, it should rotate easily!.

My Arduino sketch used to work, but doesn't compile now!

Some changes have been made to support a broader range of Arduino-like boards. Older code will require updating. Fortunately it's just a few lines around the global declarations and the `setup()` function.

Old syntax:

```
Adafruit_Thermal printer(RX_PIN, TX_PIN);
```

New syntax: declare a SoftwareSerial object and pass its address to the Adafruit_Thermal constructor, like so:

```
SoftwareSerial mySerial(RX_PIN, TX_PIN);
```

```
Adafruit_Thermal printer(&mySerial);
```

Then, in the setup() function:

```
mySerial.begin(19200);
```

```
printer.begin();
```

Some boards (e.g. Arduino Due) have multiple hardware serial ports, and the extras might be named `Serial1`, `Serial2`, etc. These can be used by passing the address of the serial device like so:

```
Adafruit_Thermal printer(&Serial1);
```

My sketch compiles, but the output is different than before.

The printer's features and behavior have changed over various firmware releases.

First, check the `PRINTER_FIRMWARE` value in `Adafruit_Thermal.h`, make sure it matches the value at the bottom of the printer test page (hold down the paper feed button when connecting power to print out a test page).

Some features just behave a little differently among releases...barcodes, line feeds, etc....if you were previously using an older library or an older printer, you may need to tweak the code to produce the desired formatting with a current setup.

The printer is connected to a computer's USB port but not showing up as expected.

How a USB-attached printer manifests can vary by OS, USB adapter, printer firmware version and what other devices are attached. In Linux (e.g. Raspberry Pi), you might find the printer device at `/dev/serial0`, `/dev/usb/lp0`, `/dev/ttyAMA0`, `/dev/tty.USBserial0` or other names, and the digit(s) at the end might be different.

It's common to snoop around in the system's `/dev` directory, comparing with the USB cable disconnected and then connected, to find the printer's device name. The `lsusb` command also sometimes helps.

Earlier model Tiny (USB) printers might appear by one of two different device names. Switching between them is explained toward the bottom of the "Making Connections" page. Newer units always appear in the `/dev/usb` subdirectory.

Hacking!



Look at those huge, razor-sharp image prints! You want some?

The following...

- Is an undocumented printer feature and is NOT guaranteed to work.
- May require modifying your printer — a warranty-voiding operation! Continue at your own risk.

You should only attempt this if all of the following apply:

- Have first confirmed that the printer works as expected when operated through conventional procedures.
- Have a genuine performance bottleneck that cannot be adequately resolved by adjusting the printer timing and thermal settings first.
- Are comfortable opening things and soldering.

A modified printer is not eligible for a refund or exchange.

These printers have a limited serial receive buffer. Push bits to the printer faster than it can physically heat dots and feed paper, and you experience an “overflow” — bitmap images become garbled, text and formatting commands may be skipped.

The thermal printer library tries to throttle data to the printer at just the right rate. Too fast and an overflow occurs. Too slow and it wastes your time; the printer isn't operating at peak throughput. This is an imperfect process...though we use very conservative timing estimates, the actual speed through the printer is impossible to predict...sometimes overflows still occur.

Hardware handshaking is a means by which a printer or other device can report to the microcontroller that it's ready to receive more data, virtually eliminating buffer overflows while operating at peak throughput...the paper feed stops only when it physically absolutely must. Optimal performance.

It appears that some varieties of these thermal printers support hardware handshaking (e.g. firmware v2.64, 2.68). This is barely mentioned in the datasheet, and in fact there isn't even a physical connection for this on the outside of the printer. A little surgery is in order...

The “Tiny” and “Nano” printers (plus the “Printer Guts”) have an exposed DTR pin — no modification is necessary! Scroll down to the “Code Changes” section to continue.

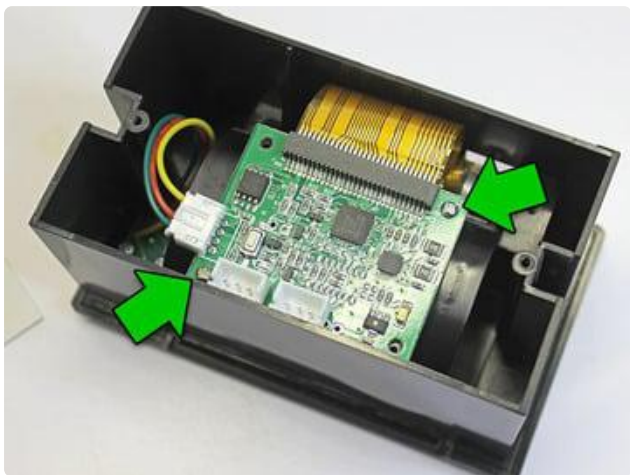
Parts and Tools Needed

- Small Phillips head screwdriver
- Pliers
- Soldering iron and related paraphernalia
- A bit of wire...but ideally a female jumper wire

Procedure



Unplug all cables, turn the printer over and remove the two small Phillips screws.



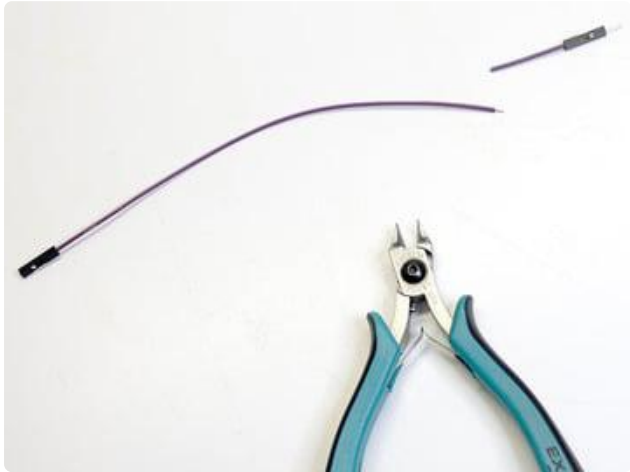
Take the back plate off, then remove the two (or sometimes four) Phillips screws holding the circuit board in place.

These screws are a little smaller than the back-holding ones...don't get them mixed up!



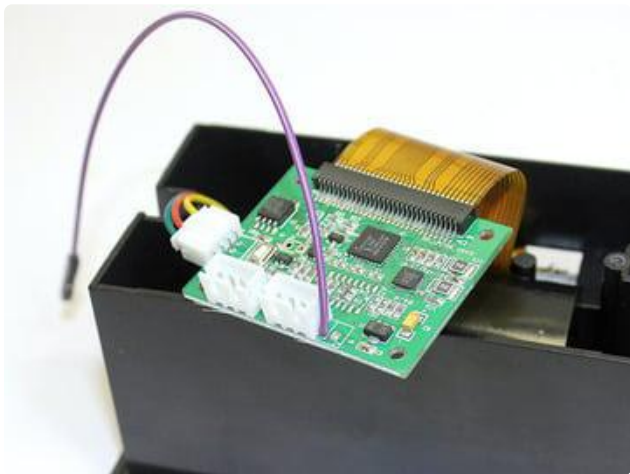
Carefully, so as not to unseat or unplug the connectors, turn the circuit board over and look for the unpopulated via labeled "DTR."

There are some other interesting solder points in here, if you're so inclined. "HV" is the raw 5–9 Volts from the power supply. On the right is a 3.3V pin, though I don't know the available current. Conceivably one could bring these out to reduce overall cabling in a project...or even install a tiny microcontroller right inside!



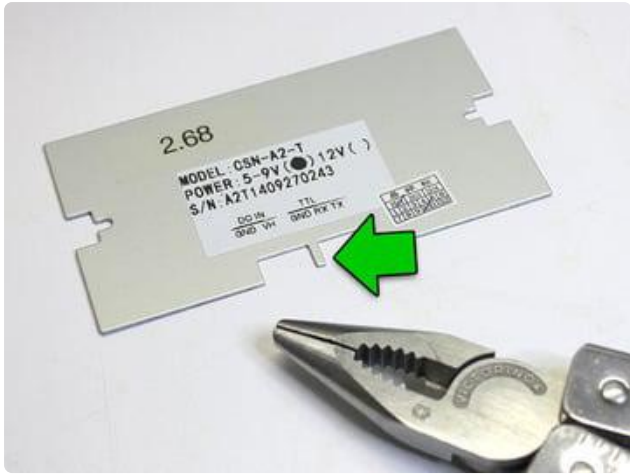
Cut an end off a female jumper wire and strip & tin the end.

This will be hanging out of the printer...so a female jumper prevents accidental contact with things if you're not using the connection. If you only have regular wire, that's fine, just be careful not to leave a bare end dangling.

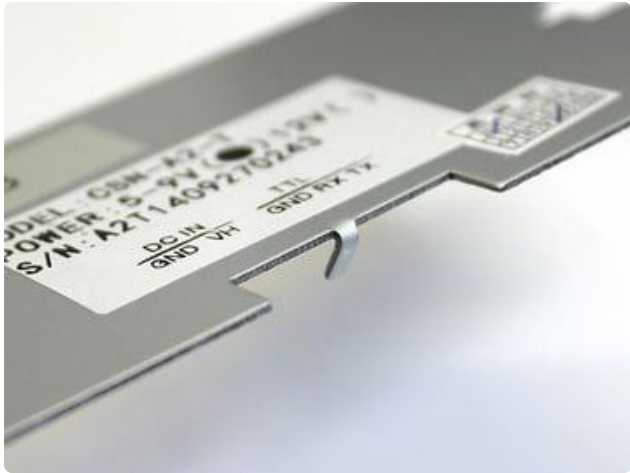


Solder the wire to the DTR pad. Top, bottom, doesn't matter...it's right up against the serial connection plug, so use whatever path works best for you, there's ample room for routing the wire around either way.

Language pedants may note that this isn't technically a DTR pin, but rather CTS. It's long-standing thing among printer manufacturers...apparently the misnomer was made decades ago but has stuck for consistency.



On the back plate, there's a small metal "finger" between the serial and power sockets. Using pliers, this can be bent back to provide an exit route for the DTR wire.



Screw the controller board back in place (check that neither of the cables has come unseated), routing the DTR wire around between the two sockets, then screw the back on.

Finished with the hardware!

You can then reconnect the power and serial sockets, and wire those up as before.

Use a jumper wire to connect DTR to any available Arduino pin. In our examples, we'll use digital pin 4.

The printer electronics operate at 3.3V (but are "5V safe"), so no level shifting is needed with 3.3V boards (Arduino Due, etc.)...this can safely be connected directly.

Code Changes

Just one line...the `Adafruit_Thermal` constructor...needs changing. It can accept an optional parameter, a pin number to use for DTR:

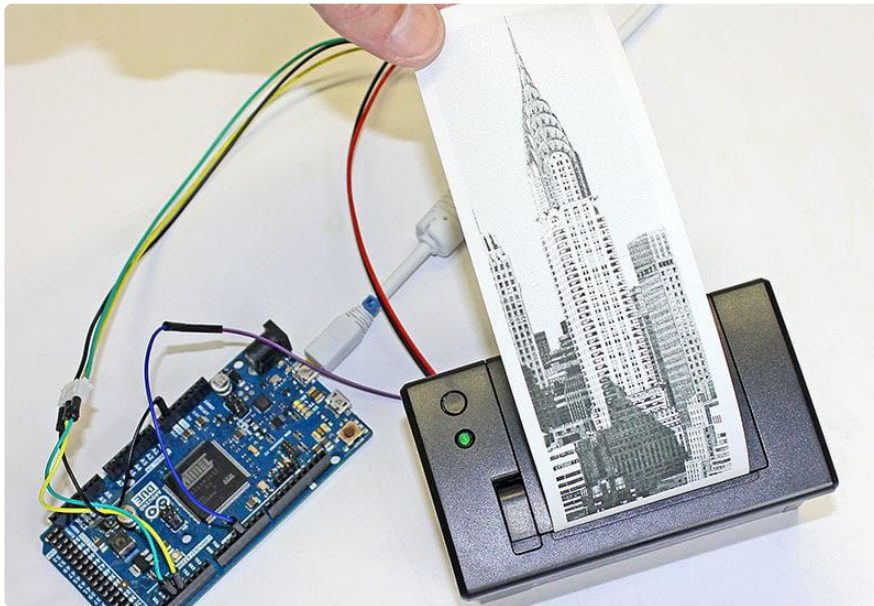
```
Adafruit_Thermal printer(&mySerial, 4);
```

This works just as well with a hardware serial port (e.g. Arduino Mega or Due):

```
Adafruit_Thermal printer(&Serial1, 4);
```

No other changes are necessary. Try this with one of the example sketches...you'll find the printer is suddenly lots faster! That's because the software throttle is no longer used...the printer continually reports its actual "ready" state to the microcontroller.

Printing Huge Images



The `printBitmap()` function can output images from an open stream or stored in `PROGMEM` (flash memory)...as explained on the "Printing Bitmaps" page.

Although the Arduino Mega has a whopping 256K flash space, a limitation of the AVR microcontroller is that a single array can't exceed 32K...that's about a 384x680 pixel bitmap image. If you try to embed a larger image in your code, the compiler will report an error.

One workaround might be to break really long images into multiple smaller images, and print these out consecutively without a feed() in between.

Another is to use a non-AVR Arduino-compatible board, such as the 32-bit Arduino Due. This has no problem with massive arrays. The Chrysler Building image above is 384x1132 pixels!

Other Things to Know

This type of printer fares best with light line art and sometimes dithered photographic images as long as the overall dot density is fairly low, like under 50%. Large solid-filled areas exhibit strange streaky artifacts...this isn't a bug of the library or printer firmware, but just a side-effect of how receipt printers operate, that they can only heat so many dots at a time and have to pull shenanigans to go beyond that, else they jam.



Here are a couple examples from fancy commercial receipt printers.

Notice in the first one that the “solid black” area isn't really solid black...examining it closely, you can see it's densely dithered, but not 100% filled.



The second does have solid fills, but limits the total area. On any given row, only so many pixels are set.

If you try to print a “dense” image and the paper jams (image gets squashed vertically), pass a lower density value to `printer.begin()`. Default value is 120. So for example:

```
printer.begin(80);
```

DTR support is not a panacea. Glitches occasionally do still happen...sometimes overflows, sometimes “framing errors” with serial data. But overall it seems fairly reliable and buttery smooth!