

Pocket AVR Programmer Hookup Guide a learn.sparkfun.com tutorial

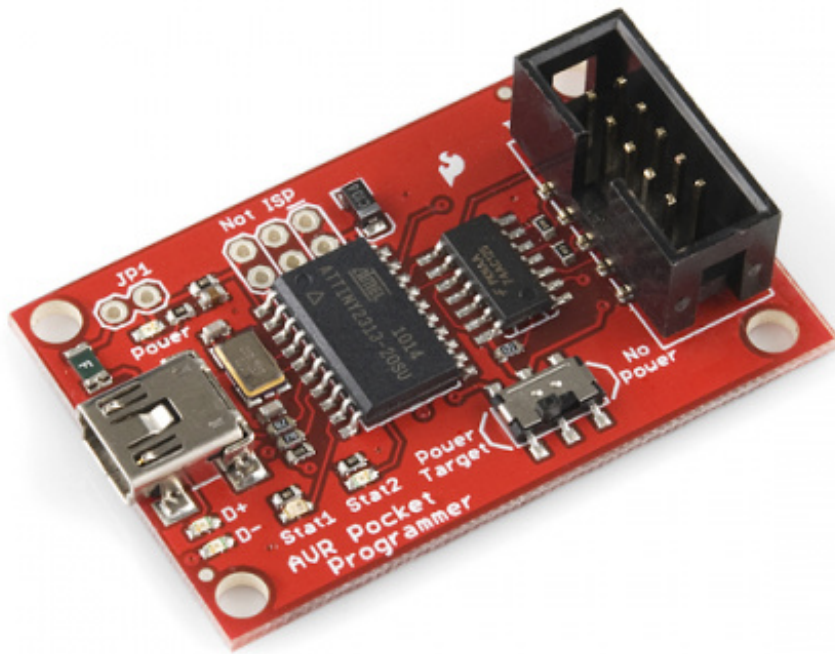
Available online at: <http://sfe.io/t214>

Contents

- [Introduction](#)
- [Board Overview](#)
- [Installing Drivers](#)
- [Programming via Arduino](#)
- [Using AVRDUDE via Command Line](#)
- [Troubleshooting](#)
- [Resources and Going Further](#)

Introduction

Do you need more control over your AVR's? Whether it's an [ATmega328](#), [ATmega32U4](#), [ATtiny85](#), if it's an AVR there's a good chance the [AVR Pocket Programmer](#) can program it.



[Pocket AVR Programmer](#)

PGM-09825

\$16.95

53

[Favorited Favorite](#) 28

[Wish List](#)

There are many reasons for programming your AVR via an in-system programmer (ISP). If your AVR doesn't have a bootloader on it, it's probably the only way to load code. Or maybe you want to overwrite the bootloader to squeeze out some extra flash space. Or maybe you want to poke at the fuse bits, to change the brown-out voltage. Or maybe you just want a faster and more reliable code upload.

Covered In This Tutorial

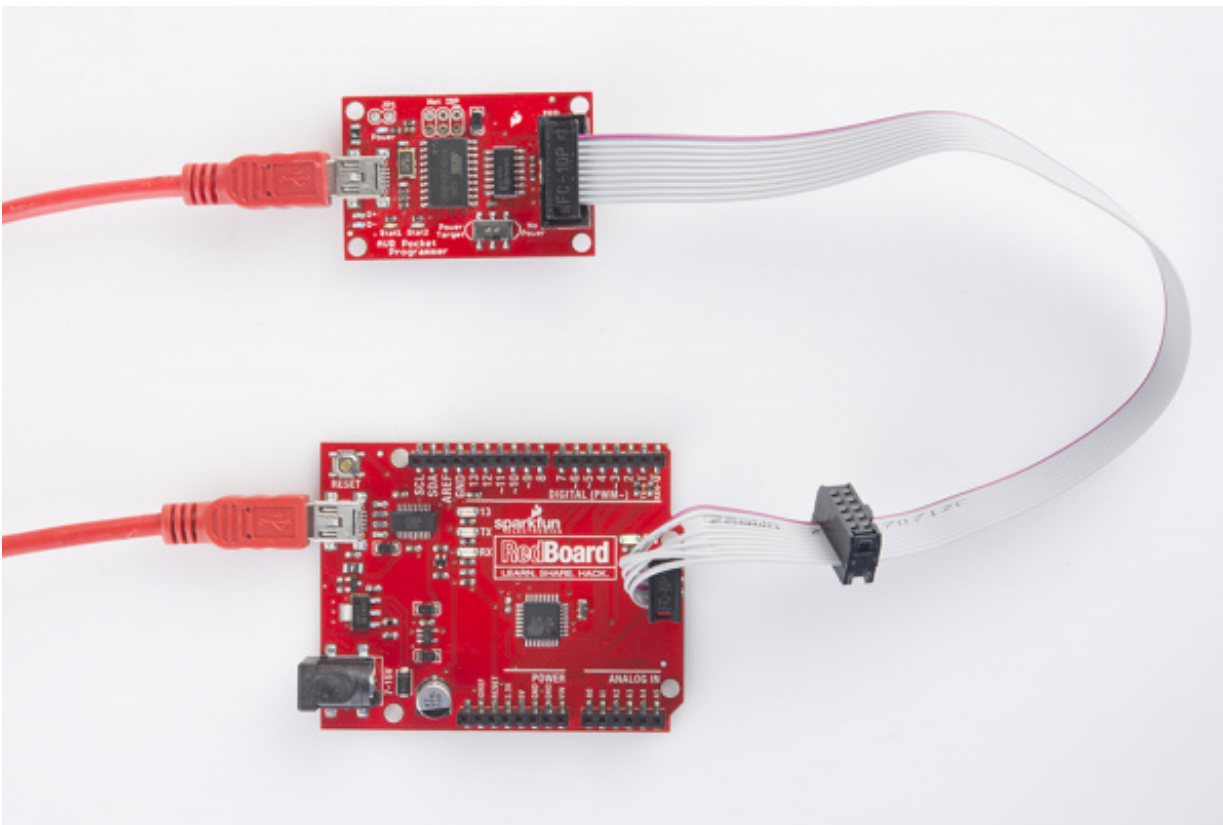
In this tutorial we will introduce you to all of the important aspects of the AVR Pocket Programmer. It's split into a series of sections, which cover:

- [Board Overview](#) -- A look at the hardware components that make up the AVR Pocket Programmer.

- [Installing Drivers](#) -- How to install the AVR Pocket Programmers on a Windows machine(*Mac and Linux users can skip this page*).
- [Programming via Arduino](#) -- How to use the ubiquitous "easy-mode" AVR IDE to upload sketches via the AVR Pocket Programmer.
- [Using AVRDUDE via Command Line](#) -- A more advanced, command-line-based approach to using the AVR Pocket Programmer.
- [Troubleshooting](#) -- A few troubleshooting tips for resolving some of the AVRDUDE errors that you may run into.

Required Materials

Most importantly, to follow along with this tutorial, you will need an [AVR Pocket Programmer](#) and an **AVR to program**. On top of that, a [mini-B USB cable](#) is required to connect the Programmer to your computer.



That microcontroller-to-be-programmed can be any AVR with **64K or less of flash**. The ATmega328 on an [Arduino Uno](#) or [RedBoard](#) works perfectly, but the ATmega2560 of an Arduino Mega *does not*.

Beyond that, you may need something to interface the Programmer to your AVR. Here are some **useful accessories**, which might make the job easier:

- [Straight Male Headers](#) -- If you have an AVR on a development board -- like an [Arduino Pro](#) -- the 2x3 (or 2x5) ISP header may not be populated. You can use straight male headers (also available in a [long-pinned version](#)) to make a temporary contact between ISP cable and your dev board. There is also a [2x3 pin version](#).

- [ISP Pogo Adapter](#) -- Like the headers, this ISP adapter is designed to provide a temporary electrical connection between adapter and AVR. This is a great, more reliable alternative to the headers.



[Break Away Headers - Straight](#)

PRT-00116

\$1.50

20

[Favorited Favorite](#) 121

[Wish List](#)



[SparkFun ISP Pogo Adapter](#)

KIT-11591

\$12.95

4

[Favorited Favorite](#) 11

[Wish List](#)



[Break Away Headers - 40-pin Male \(Long Centered, PTH, 0.1"\)](#)

PRT-12693

\$0.75

1

[Favorited Favorite](#) 15

[Wish List](#)



[Header - 2x3 \(Male, 0.1"\)](#)

PRT-12807

\$0.50

[Favorited Favorite](#) 4

[Wish List](#)

Note: If your AVR is living on a breadboard, you probably don't have an interface to the standard 2x3 ISP pinout. Our old [simple breakout board](#) made interfacing the programmer with your breadboarded circuit possible. We recommend using the ISP Pogo Adapter linked above now that the breakout board has been retired in the catalog.

Suggested Reading

Whether you're a beginner or experienced electronics enthusiast, the Pocket Programmer should be easy to get up-and-running. If you've programmed an Arduino before, you'll be well-prepared for the next step. Here are some tutorials we'd recommend reading before continuing on with this one:

- [What is an Arduino?](#) -- If you're unfamiliar with AVRs, check out this tutorial to learn about the most popular one of the lot.
- [Installing Arduino](#) -- Arduino isn't required to use the Programmer, but it can make things easier, especially if you still want to program your AVR using the Arduino libraries.
- [Serial Peripheral Interface \(SPI\)](#) -- The Pocket Programmer uses an SPI interface to send data to and from the AVR. Click this tutorial to learn the meanings behind "MOSI", "MISO", and "SCK".

[Serial Peripheral Interface \(SPI\)](#)

SPI is commonly used to connect microcontrollers to peripherals such as sensors, shift registers, and SD cards.

[Favorited Favorite](#) 84

[What is an Arduino?](#)

What is this 'Arduino' thing anyway? This tutorial dives into what an Arduino is and along with Arduino projects and widgets.

[Favorited Favorite](#) 44

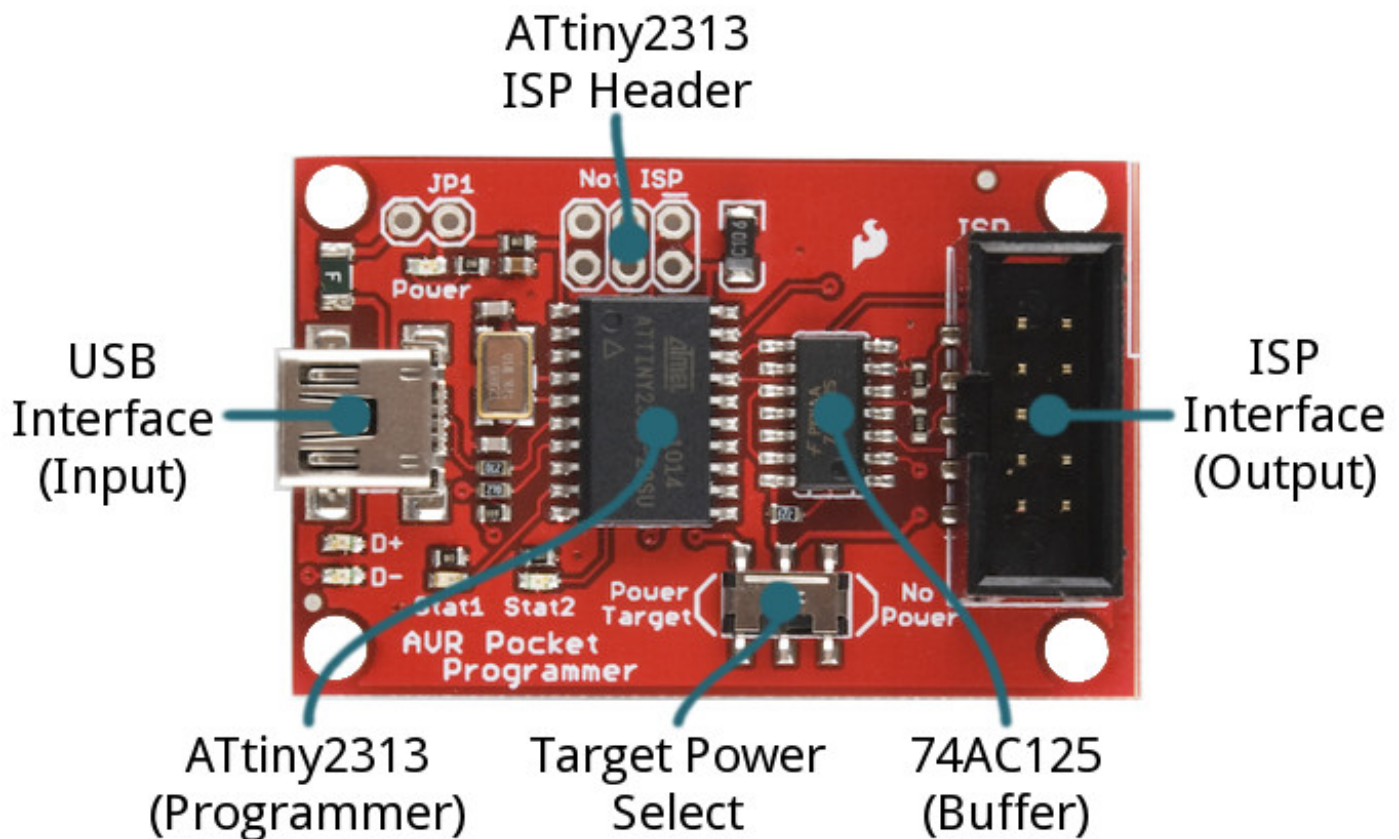
[Installing Arduino IDE](#)

A step-by-step guide to installing and testing the Arduino software on Windows, Mac, and Linux.

[Favorited Favorite](#) 16

Board Overview

Before we get to using the AVR Pocket Programmer, let's quickly overview what components fill the board out:



- **USB Connector** -- This is your **data and power input** to the Programmer. A [mini-B USB cable](#) plugs in here and connects your computer to the Programmer.
- **2x5 ISP Header** -- This shrouded header mates with the included [Programming Cable](#), and allows you to send the programming signals out to your AVR. It's polarized to make sure you can't plug anything in backwards.
- **Power Target Switch** -- Unlike a lot of ISP's out there, the AVR Pocket Programmer can **deliver power** to the AVR-to-be-programmed. Flick this switch to the "Power Target" side, to send 5V to the AVR. More on this below.
- **ATtiny2313** -- This is the chip that works the programming magic. It converts between USB

and SPI to turn commands from your computer into words and instructions to load into your AVR-to-be-programmed. Unless you want to customize the Tiny ISP firmware, you can **leave this chip alone**.

- The unpopulated ISP header, above the ATtiny2313, is broken out in case that chip needs to be programmed. It's mostly used in production by those [who program the programmers](#).
- **74AC125 Buffer** -- This chip helps to add some protection to the programmer by buffering the data-line outputs. Another IC to mostly ignore.

The board also includes a variety of LEDs to indicate power, status, and data transfers.

AVR ISP Pinouts

AVRs are programmed through an [SPI interface](#). There are six unique signals required for communication between ISP and AVR: VCC, GND, Reset, MOSI, MISO, and SCK.

To route those signals between devices, there are two standardized connectors -- one 10-pin, 2x5 and another 6-pin, 2x3 connector:

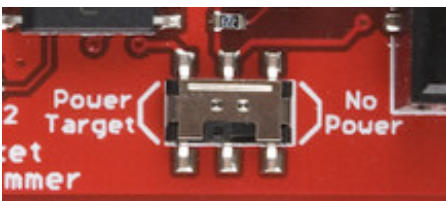


AVR ISP pinouts -- top view.

The AVR Pocket Programmer includes an on-board 2x5 connector, and the included [AVR Programming Cable](#) terminates with both 2x5 and 2x3 connectors.

Power Target Switch

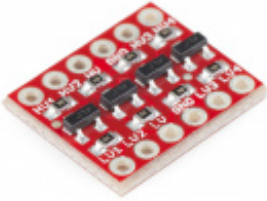
If you're working with an AVR on a breadboard or a prototype, power may be hard to come by. The AVR Pocket Programmer allows you to route **5V** out to your AVR. It can deliver upwards of **500mA** before tripping the onboard PTC.



If the switch is in the *Power Target* position, it will route 5V out to your AVR. Otherwise, if the switch

is pointing towards *No Power*, no signal will be connected to the 5V pin on the ISP connector.

⚡ **Warning!** Be careful using this feature! It will **output 5V and only 5V!** If you're working with a 3.3V or 1.8V system, make sure this switch is in the *No Power* position and use a logic level converter.



[SparkFun Logic Level Converter - Bi-Directional](#)

BOB-12009

\$2.95

111

[Favorited Favorite](#) 138

[Wish List](#)

Installing Drivers

Driver installation is required on **Windows machines only**. If you're using a **Mac or Linux** machine, you **don't need to install drivers**. Just plug the board in, and skip to the [next section](#). Otherwise, follow along below as we overview the installation process.

[Pocket AVR Programmer Hookup Guide - Programming via Arduino](#)

There are two sets of instruction for driver installation on this page. The [first is the easiest, quickest method](#), and should work for most everyone. The [second installation process](#) is only required if the first one fails -- it takes a more manual approach to the driver installation.

[Automatically Install the Drivers with Zadig](#)

To begin, **plug the AVR Pocket Programmer into your computer**. Upon initially connecting the board, Windows will try to automatically install the drivers. Some computers may be lucky, but most will turn up with a message notifying you that the driver install failed.

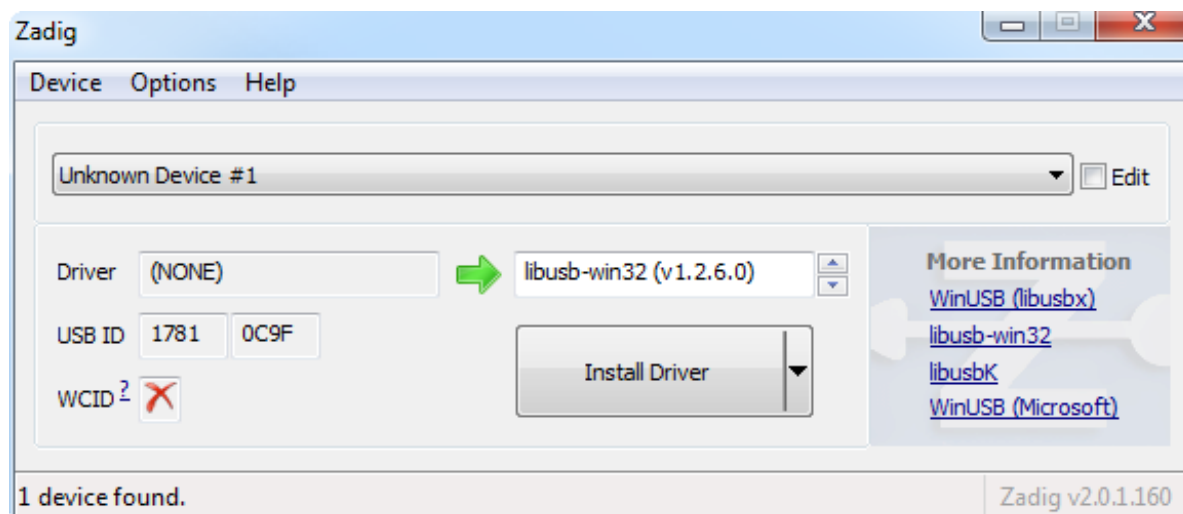
Click the link below to download the Zadig software and drivers:

[Download the Zadig USBtiny Drivers \(ZIP\)](#)

Use your favorite unzipper to extract the ZIP file. Don't forget where you put the extracted folder!

After you've plugged the Pocket AVR Programmer into your computer and your machine has run through the process of checking for and failing to install drivers, proceed to the "**zadig_v2.0.1.160**" folder you just unzipped. Then **Run zadig.exe** software.

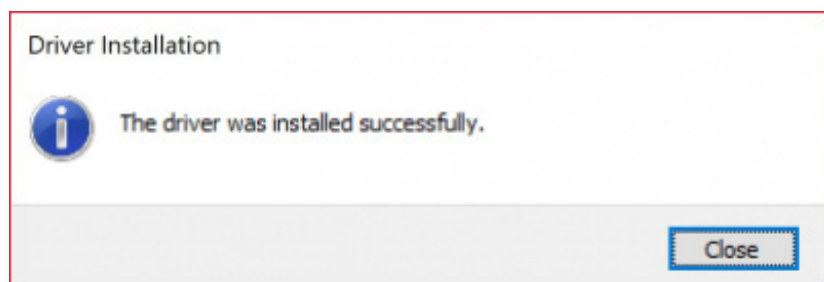
Zadig is a wonderful tool that can install the drivers on just about any Windows platform out there. Upon opening the program, you should be greeted with a window like this:



There are a few options to verify before installing the driver:

- **Select the device** -- The top dropbox controls which device you want to install the driver for. Hopefully you only have one option here, something like "**Unknown Device #1**". If you have more than one option, check your device manager to see if you can make sense of which is which (plugging and unplugging a device usually helps).
- **Select the driver** -- Click the arrows in this box until you happen upon **libusb-win32 (vx.x.x.x)**, that's the driver we want to install.

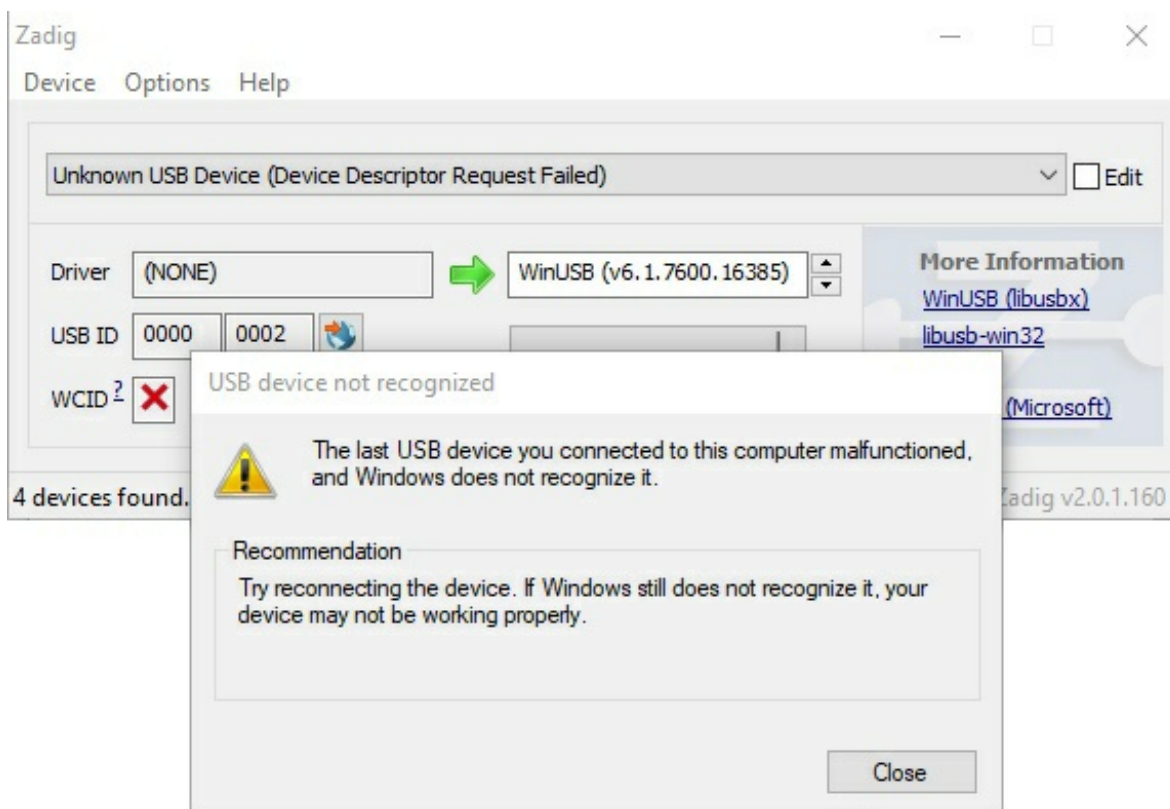
After verifying those two selections, **click "Install Driver"**. The installation process can take a few minutes, but after you've watched the scroll bar zoom by countless times, you should be greeted with a "**The driver was installed successfully**" message.



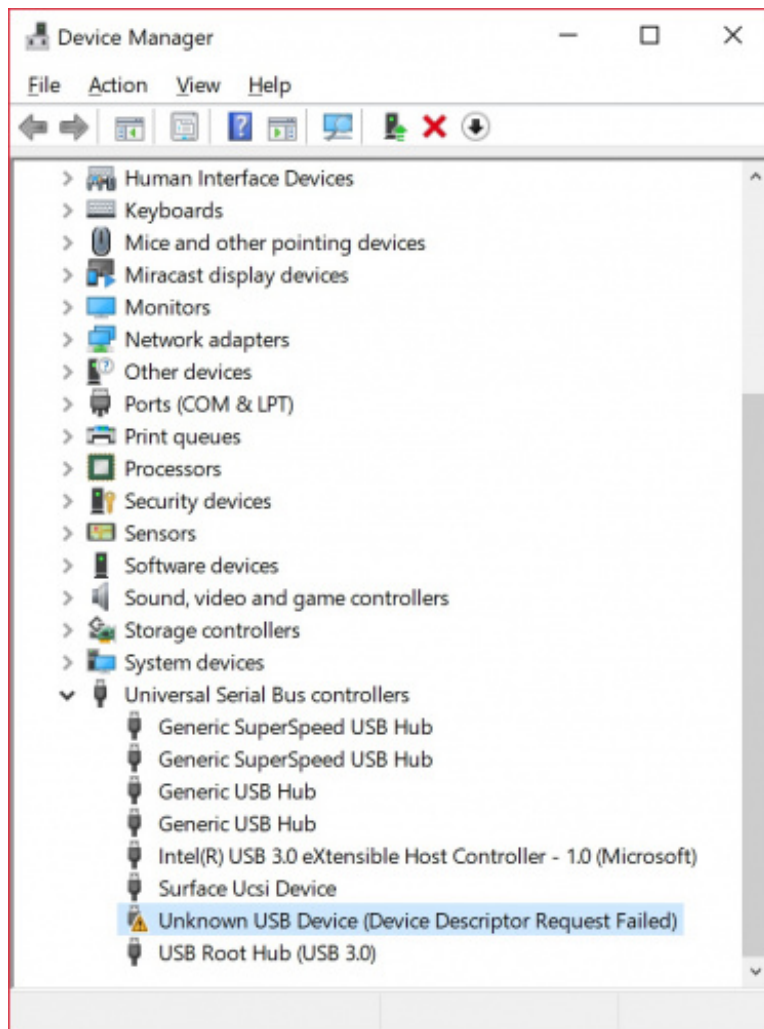
Zadig Driver Installation Issues

After installing the drivers, your computer may respond by indicating that the device was not installed correctly. Here are two methods of troubleshooting driver issues when installing with Zadig.

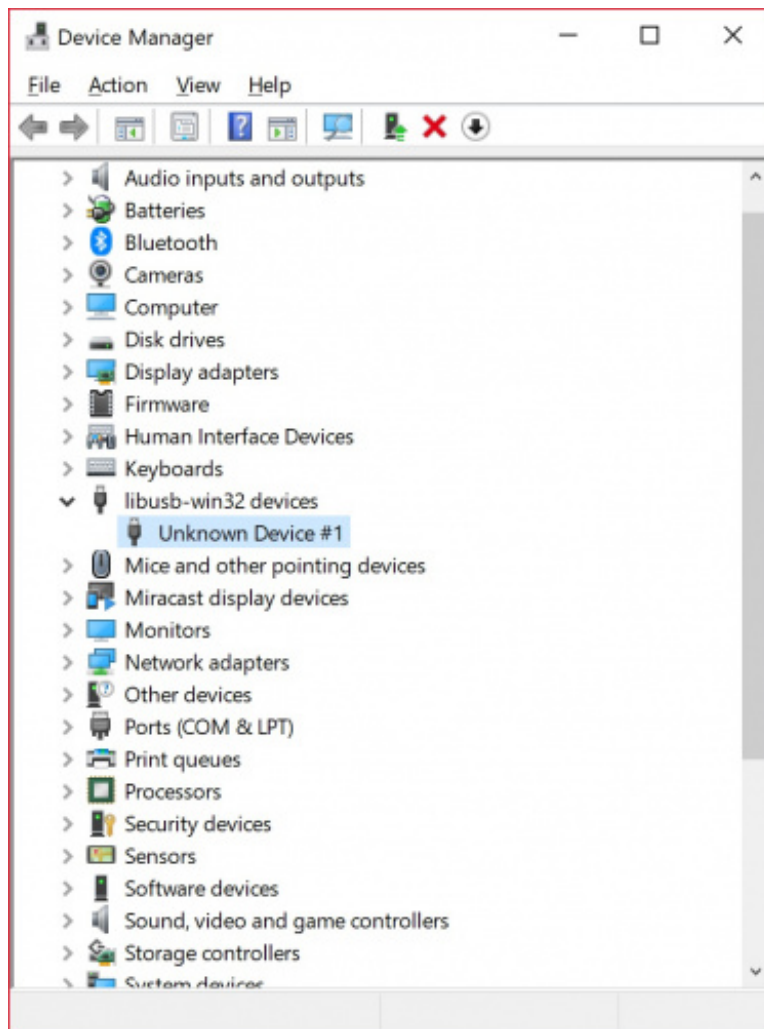
Troubleshooting Tip: In this case, the *WinUSB* drivers were selected instead of the *libusb-win32* drivers. To remedy the issue, simply go through the [guide again to reinstall the correct *libusb-win32* drivers](#).



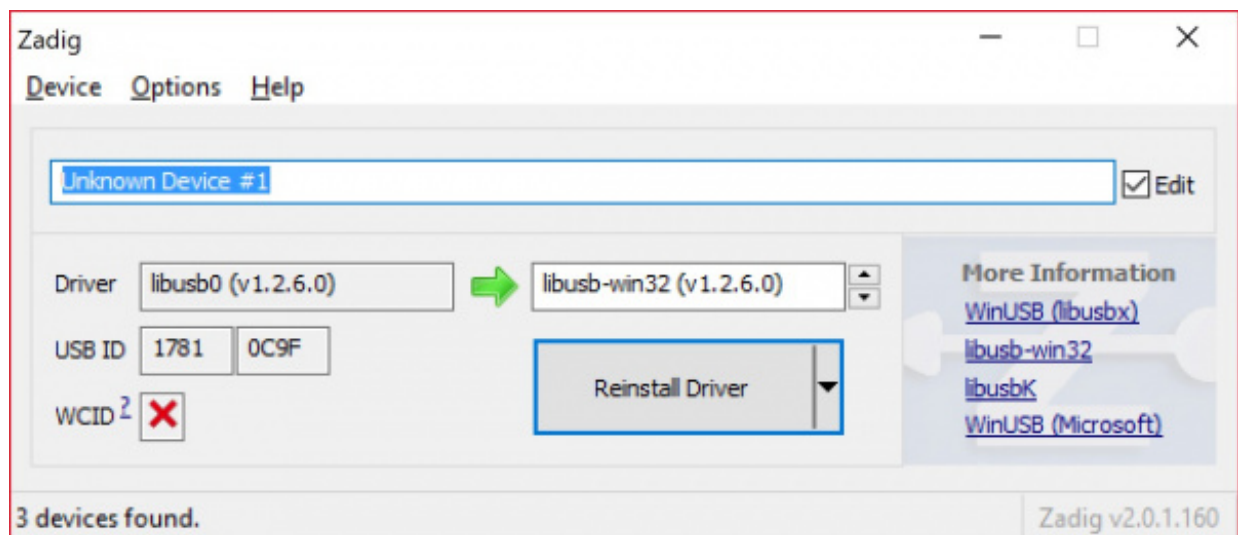
Troubleshooting Tip: In other cases, it may also initialize somewhere in your device manager as an **Unknown USB Device (Device Descriptor Request Failed)** even if you installed the correct drivers:



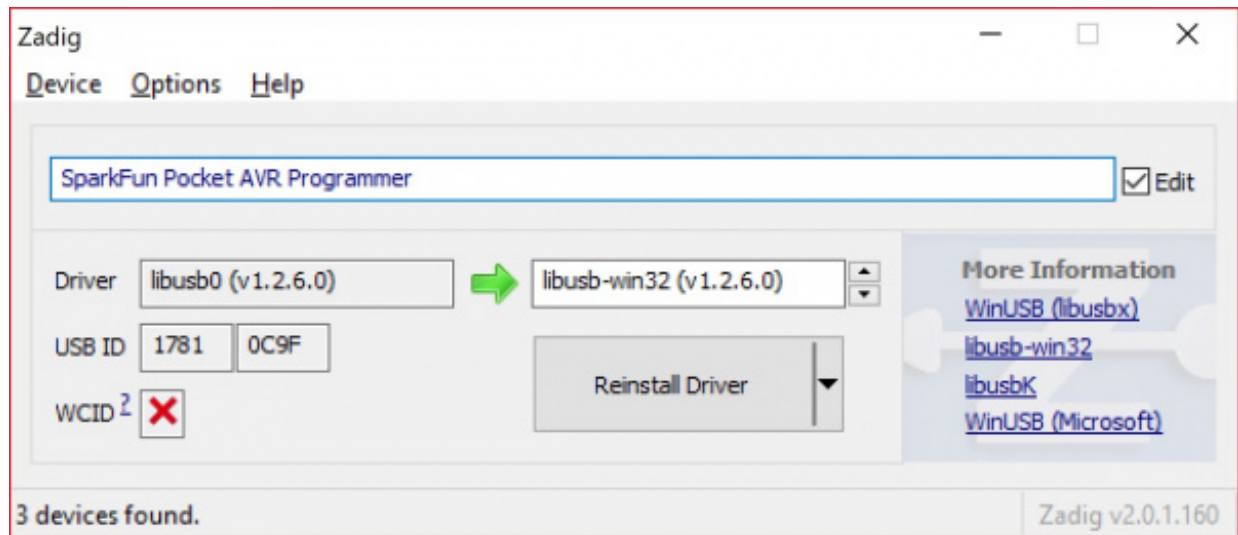
Try unplugging and replugging the Pocket AVR Programmer back into your USB port. Or switch out your mini-B USB cable for a known good. In some cases, your Pocket AVR Programmer may show up under the **libusb-win32 devices** as an **Unknown Device #1**. As long as it shows up under **libusb-win32 devices** tree, you should be good to go!



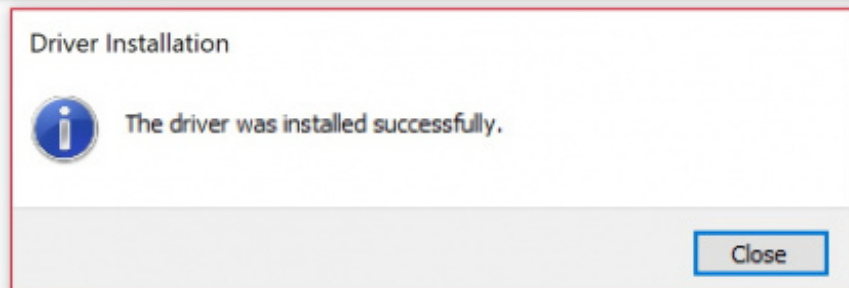
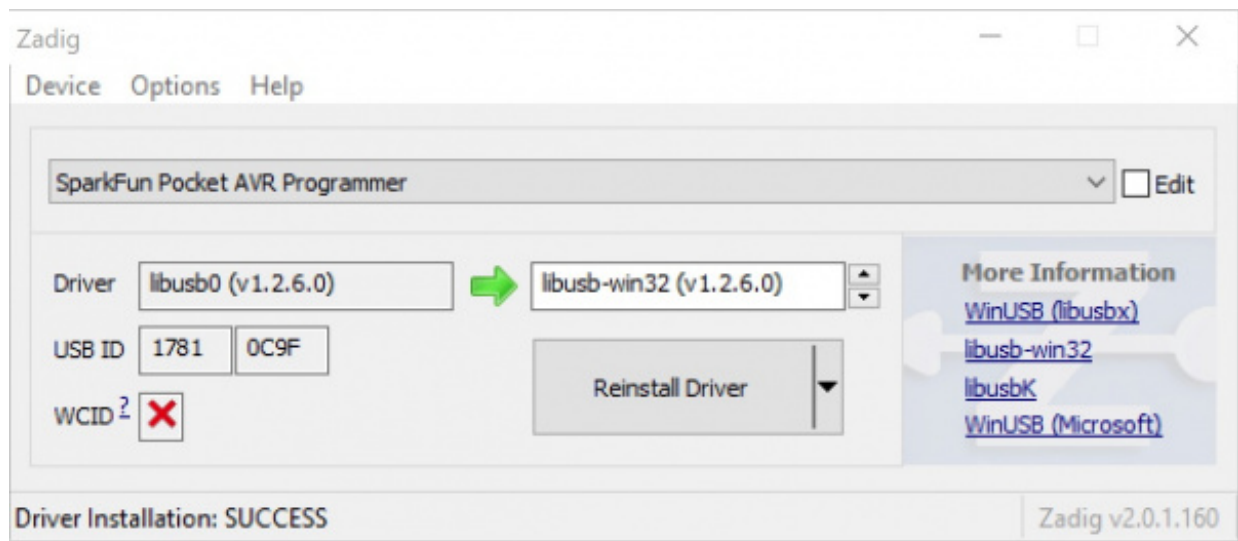
Well done! You've successfully installed the drivers on your computer. However, the driver still shows up as an **Unknown Device #1**. But you know what it is! You can use the Zadig software to rename the USB port if you desire. With your programmer connected to your computer and the software open, navigate to the programmer's port. Select the checkbox next to **Edit**.



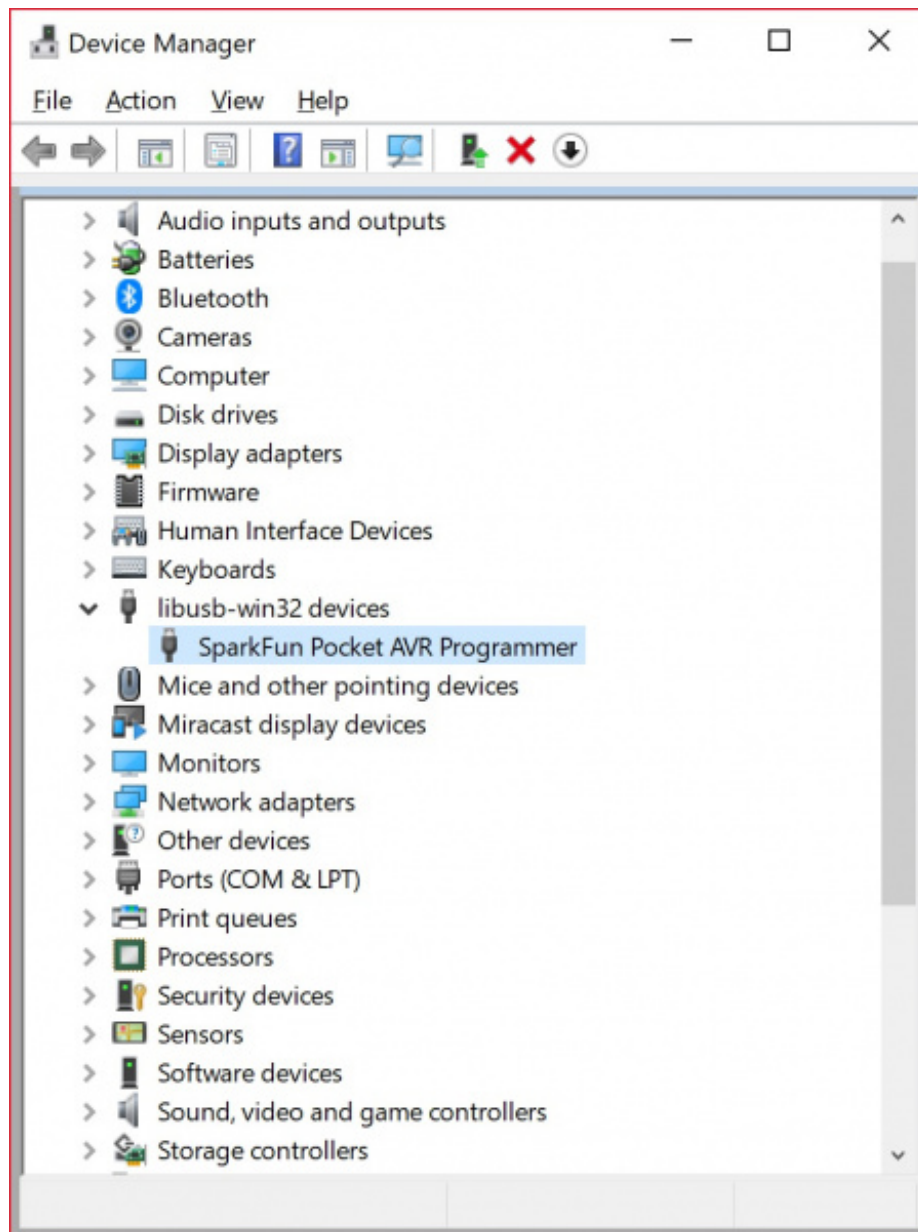
Type in the name for your port. It can be **"USBTiny"** or in this case, **"SparkFun Pocket AVR Programmer"**. Make sure that the correct driver is selected.



Click **Reinstall Driver**. The driver will reinstall and you should see the same message that indicates that the drivers were successfully installed. You may need to unplug and replug the programmer to your computer to give it a second to refresh again.



Open up your device manager and you should see the device renamed!



If you were successful, close out of the Zadig program and [proceed to the next section!](#)

[Pocket AVR Programmer Hookup Guide - Programming via Arduino](#)

If Zadig didn't work for you, check out the instructions below for help manually installing the drivers.

Manually Installing the libUSB Drivers

If, for some reason, Zadig didn't work for you. Read the instructions below to manually install the drivers. Click the link below to **download the drivers**:

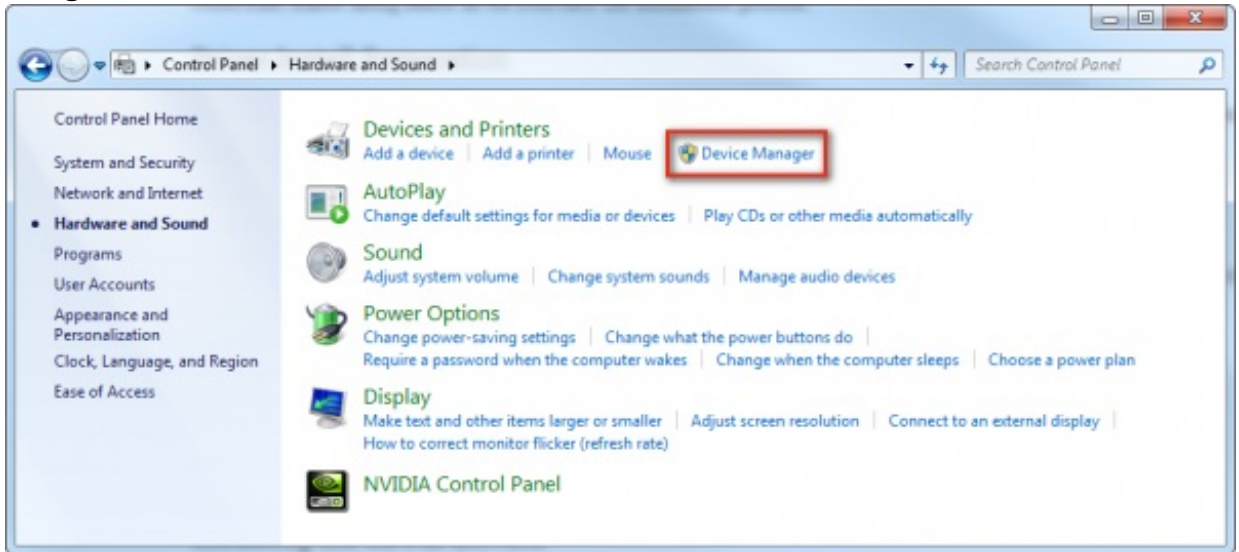
[Download the USBtiny Drivers \(ZIP\)](#)

Use your favorite unzipper to extract the ZIP file. Don't forget where you put the extracted folder!

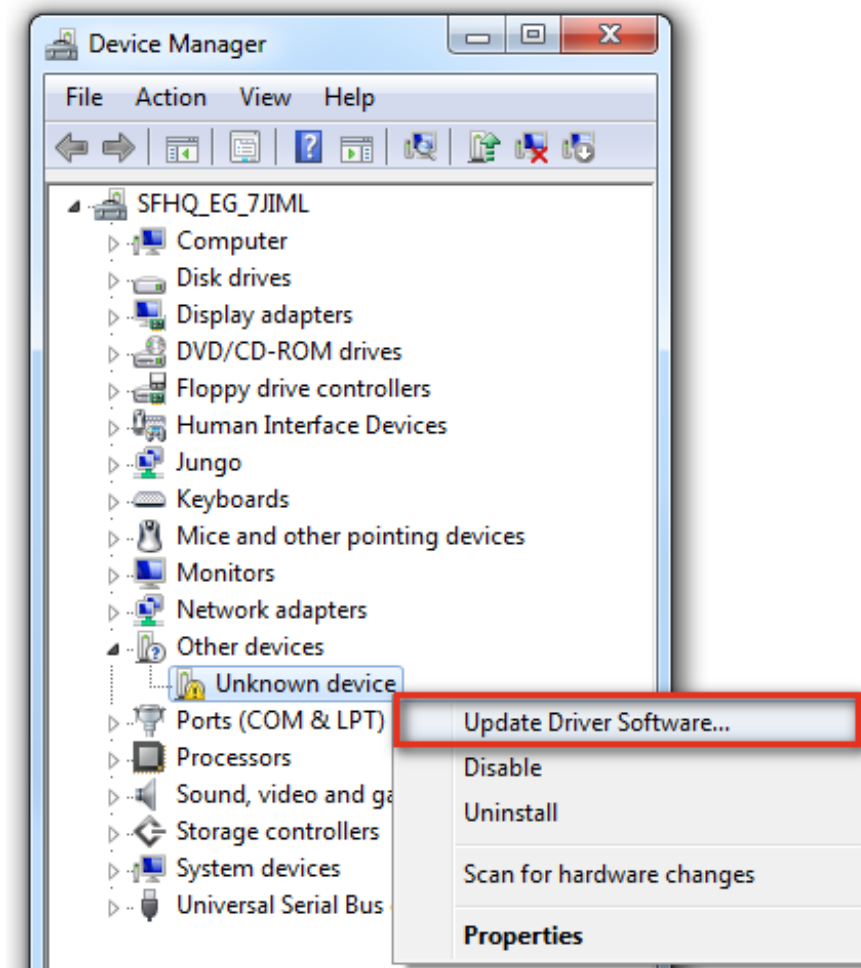
After you've plugged in the Programmer, and Windows has failed to install the driver. Follow these

steps to install the driver:

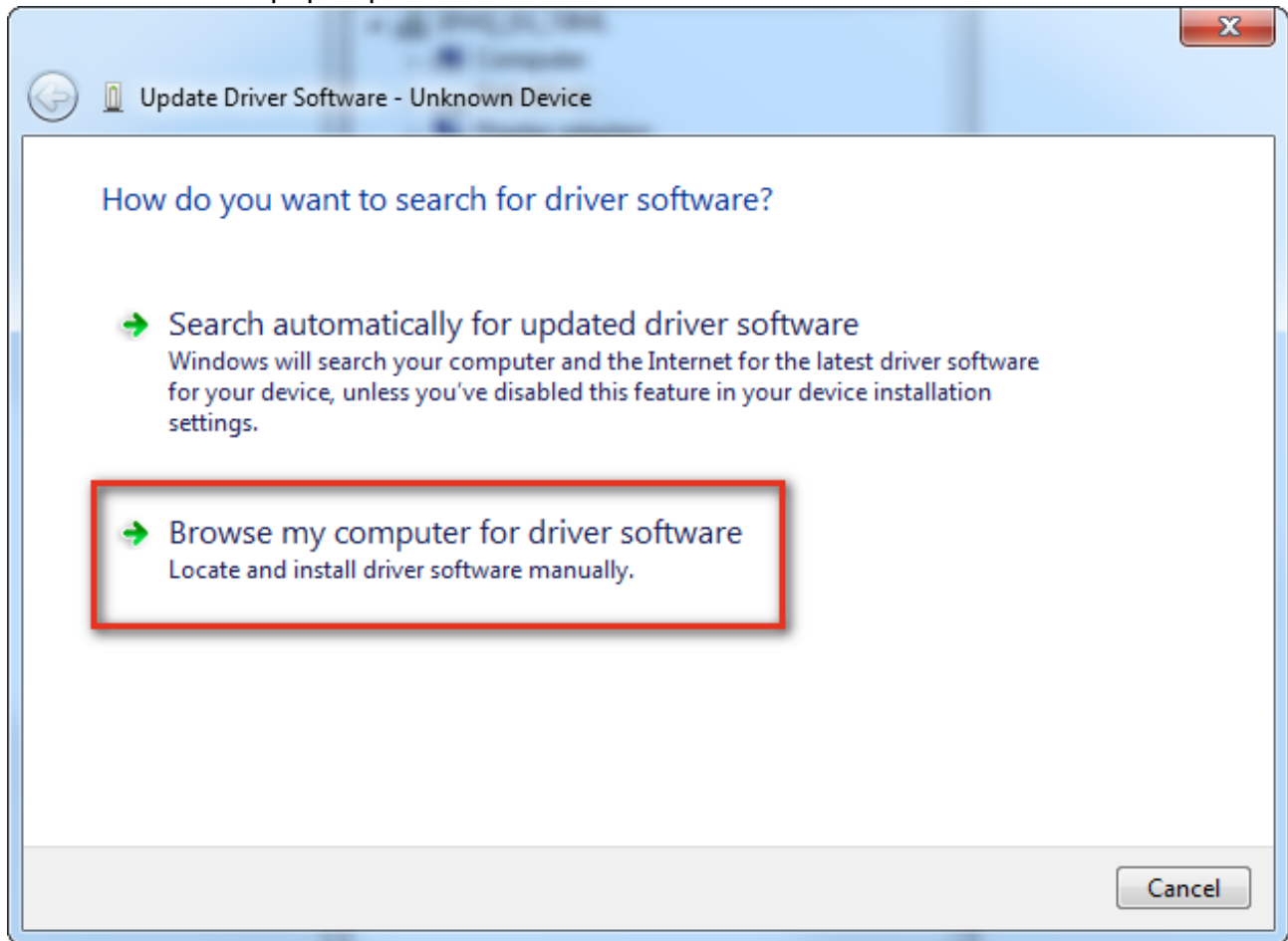
1. **Open the Device Manager** -- There are a few routes to open up the device manager.
 - You can go to the **Control Panel**, then click **Hardware and Sound**, then click **Device Manager**.



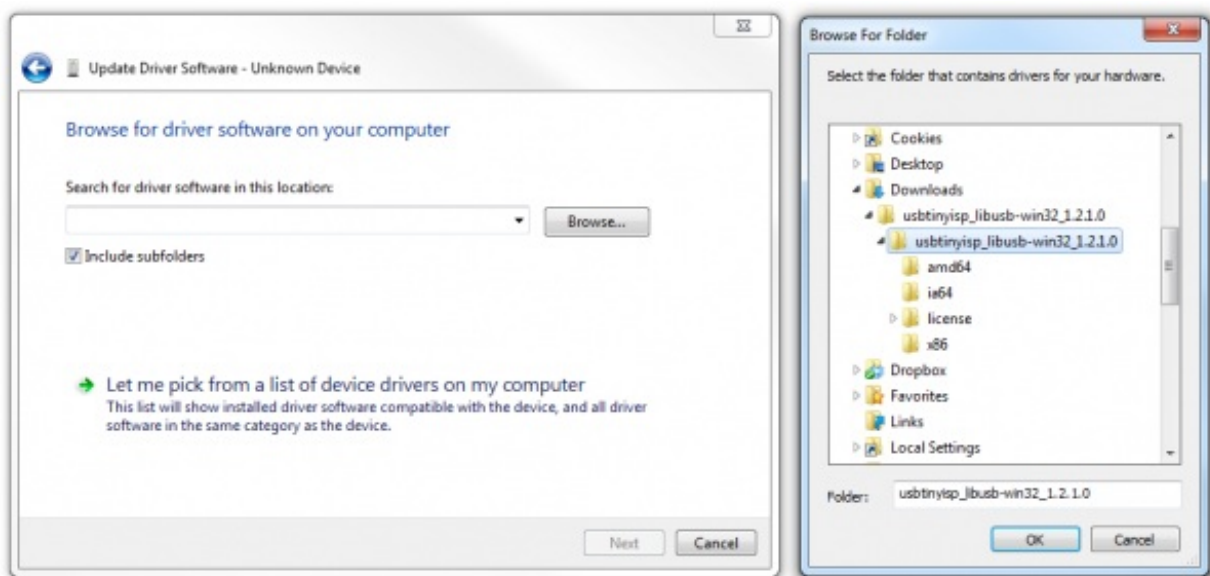
- Or, simply open the **run tool** (press Windows Key + R), and run `devmgmt.msc`.
2. In the Device Manager, you should see "**Other devices > Unknown device**". **Right click "Unknown Device"** and select **Update Driver Software...**



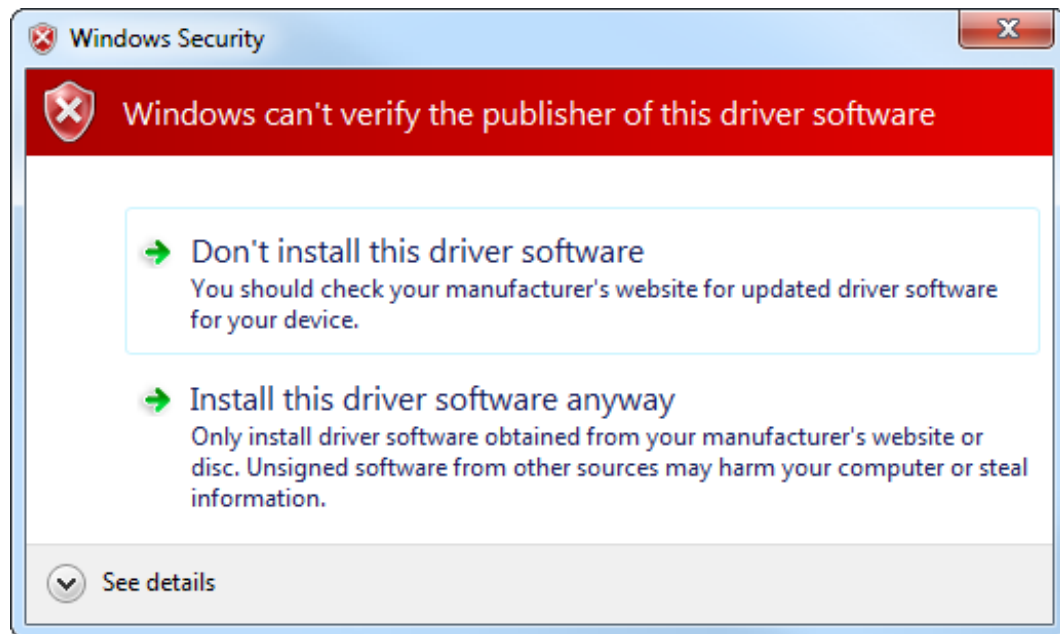
3. Click **Browse my computer for driver software** in the "Update Driver Software - Unknown Device" window that pops up.



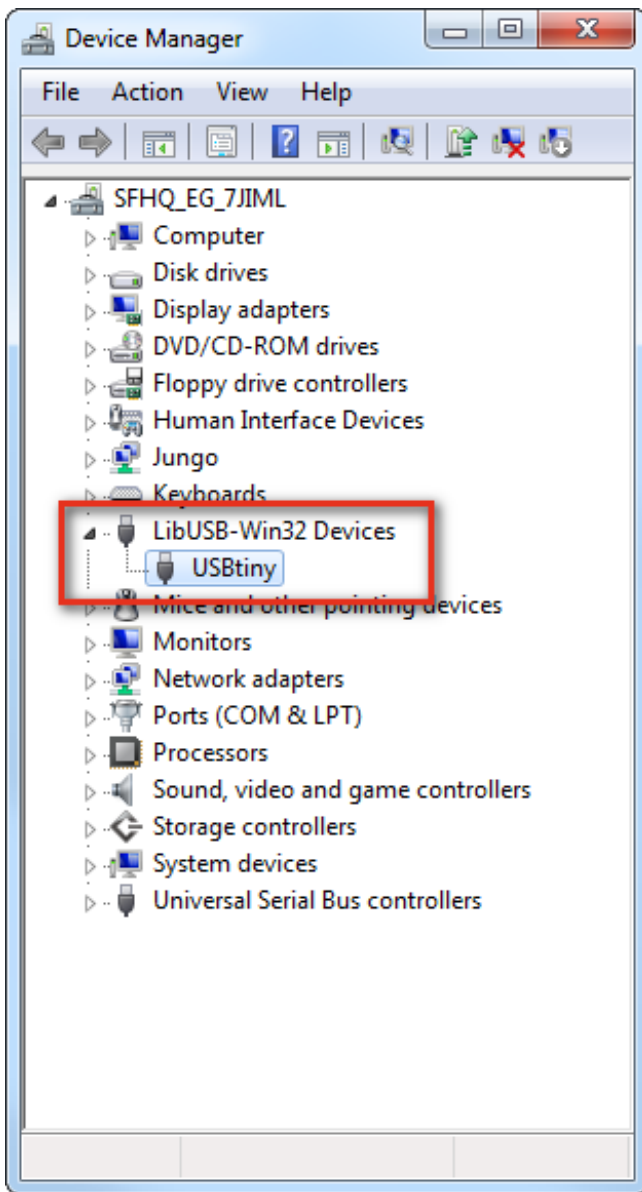
4. Click "**Browse...**" and navigate to the `../usbtinyisp_libusb-win32_1.2.1.0` folder you just downloaded. Then click **Next**.



5. Windows will begin installing the driver, and then immediately notify you that the driver isn't signed. Click **Install this driver software anyway** option, to proceed with the installation.



6. After a few moments, the driver should successfully install. You'll be prompted with a **"Windows has successfully updated your driver software"** window. Close that, and you'll see a **"USBtiny"** entry populated in the Device Manager, under the **'LibUSB-Win32 Devices'** tree.



Congratulations! [Proceed over to the next section](#), and we'll start using the Programmer!

Drivers Still Not Installing? If you are **still** having issues installing the drivers, try looking at this troubleshooting tip and driver from our technical support:

[GitHub SparkFunTechSupport: ...PGM-11801](#)

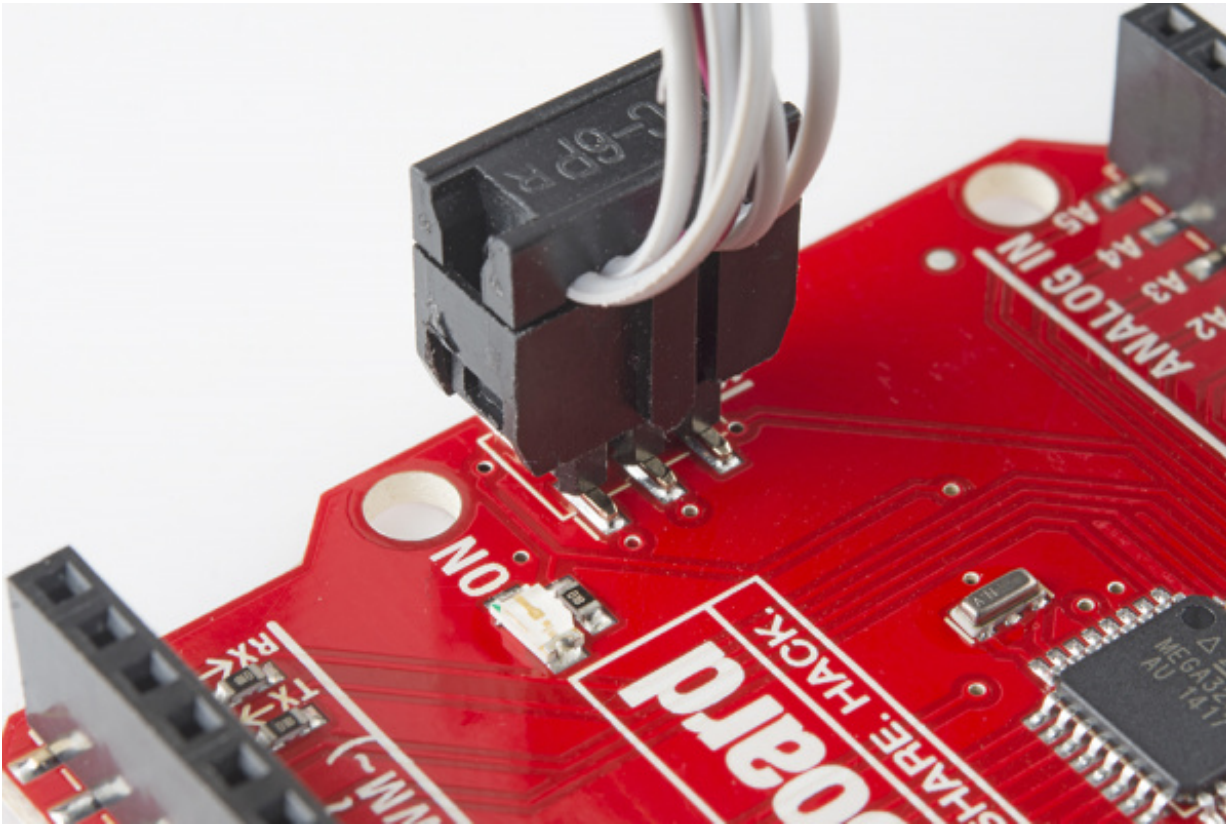
Breathe easy now! Once you've installed the USBtiny drivers on your computer, you shouldn't ever have to do it again. Now it's time to program something!

Programming via Arduino

Arduino has a built-in tool that allows you to upload your sketch via a programmer instead of the serial bootloader. If you're just taking your first steps toward ISP-ing your Arduino-compatible AVR, this is a good place to start.

Connect the Programmer

First, let's connect the programmer to our Arduino. Most Arduinos break out the standardized 2x3 ISP header towards the edge of the board. Plug the 2x5-connector end of included programming cable into your AVR Pocket Programmer, then connect the other, 2x3 end into your Arduino.



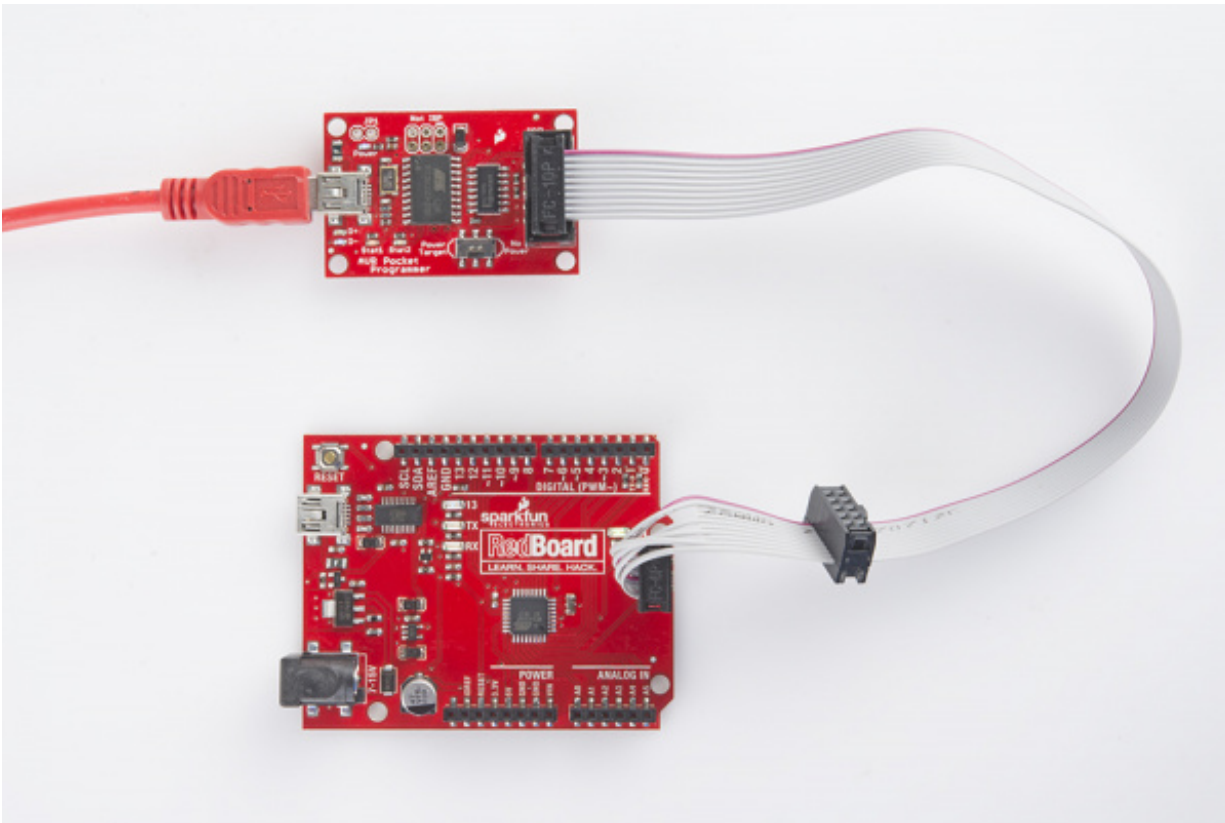
Note the notch on the connector facing the same direction as pin 1 (marked with a small white line here) on the 2x3 Arduino connector.

When connecting the programming cable to you Arduino, make sure you **match up the polarity!** The cable has a "notch" on one side of the plastic housing. This should **point towards pin 1** of the Arduino's ISP header. Pin 1 is usually indicated by a stripe next to the hole or pin.

If your Arduino doesn't have the ISP pins populated, check out the [bottom section](#) of this page for some tips and tricks we've used through the years.

Powering Target

While connecting your programmer, double-check to make sure the "Power Target" switch is in the correct position. The programmer *can* power your Arduino alone! If you want it to handle that task, slide it over to the *Power Target* position.



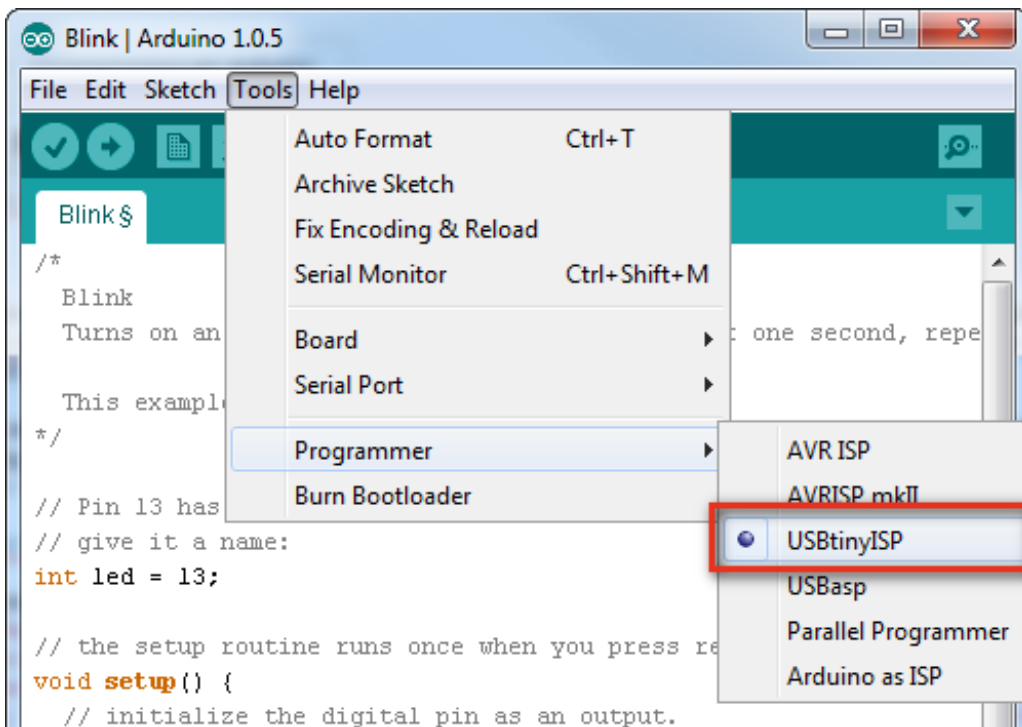
The "Power Target" feature is especially useful if you only have one USB slot/cable available.

Unplug your Arduino from USB if you're going to power it via the Programmer -- you don't want to create any ugly reverse current flows through your power sources.

Programming via Arduino

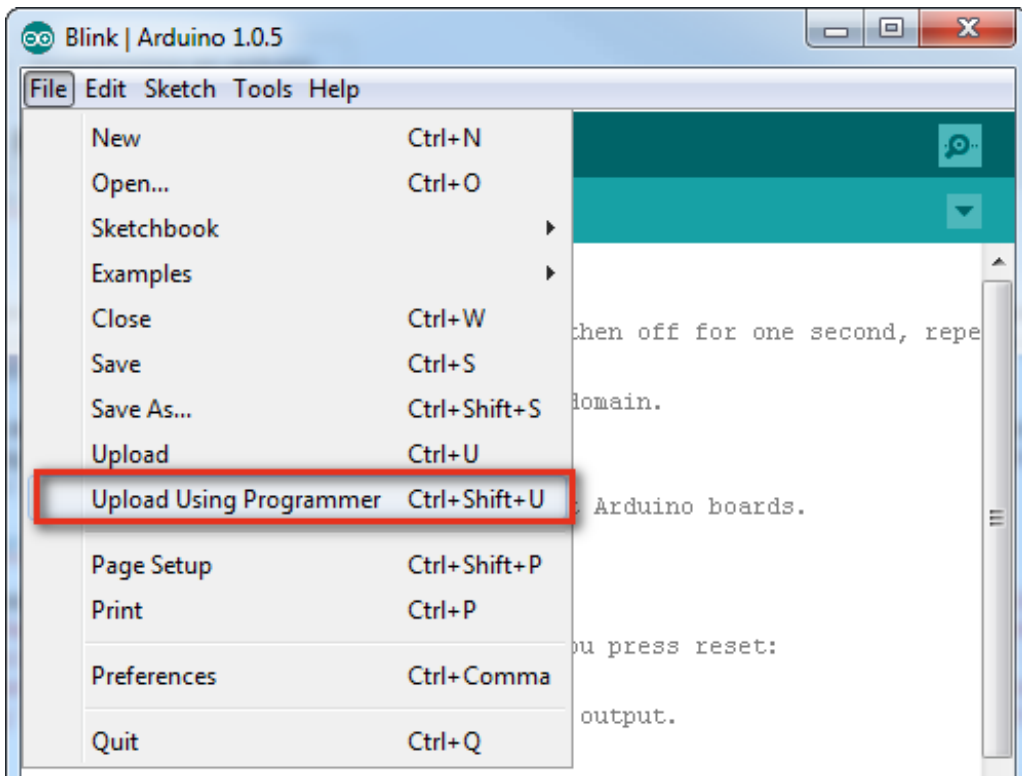
Now that the programmer is connected to your Arduino, open up the IDE. Then open an example sketch like Blink (**File > Examples > 1.Basics > Blink**).

Before uploading, we need to tell Arduino which programmer we're using. Go up to **Tools > Programmer** and select **USBtinyISP**.



Also make sure you've **set the "Board" option** correctly! The serial port selection isn't required for uploading the sketch, but is still necessary if you're doing anything with the serial monitor.

To upload the sketch using the programmer you selected, go to **File > Upload Using Programmer**. If you'll be doing this a lot, get used to pressing CTRL+SHIFT+U (COMMAND+SHIFT+U on Mac).



Note: Depending on your Arduino IDE version, this may be in a different menu. Try looking under **Sketch > Upload Using Programmer** for this option.

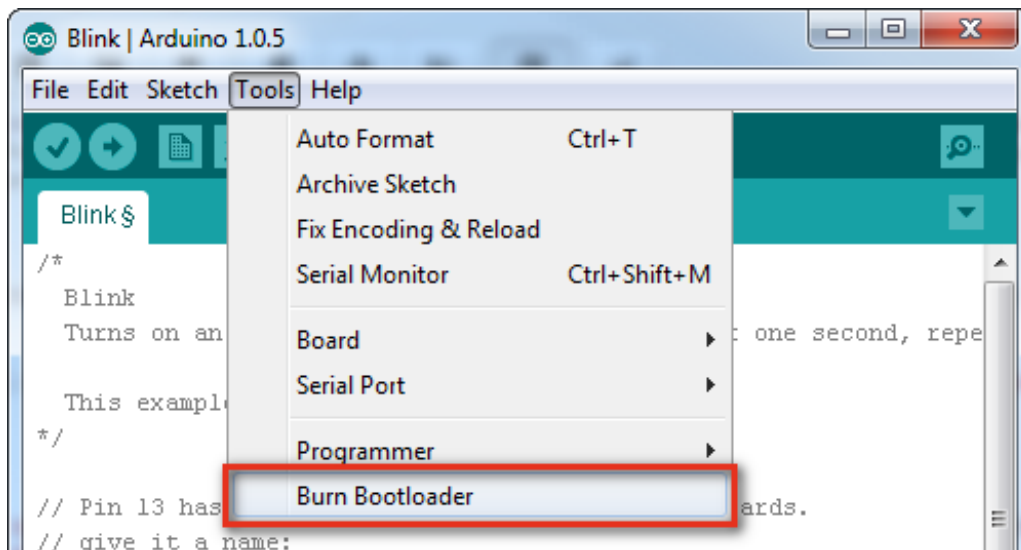
The Arduino will run through its normal process of compiling. After the sketch compiles, the Programmer will start lighting up blue everywhere -- the "D+" and "D-" LEDs will light up, and so will the "Stat2" LED. When the "Stat2" LED turns off, the upload will be finished. Check the status area of your Arduino IDE to verify that the sketch is "Done uploading."

If you've uploaded a sketch via the programmer, you've also **wiped off the bootloader**. If you ever want to put the serial bootloader back on your Arduino, check out the next section.

Programming a Bootloader

The Arduino IDE also has a feature built-in to allow you to (re-)upload a bootloader to the AVR. Here's how:

Make sure you've set the **Board** option correctly -- among other things, that will set *which* bootloader you'll be uploading. Then, simply navigate up to **Tools > Burn Bootloader** at the very bottom of the menu.



This process may take a minute-or-so. Not only will the bootloader be written into the flash of your AVR, the fuse bits (setting the clock speed, bootloader space, etc), and lock bits (barring the bootloader from overwriting itself) will also be (re)set.

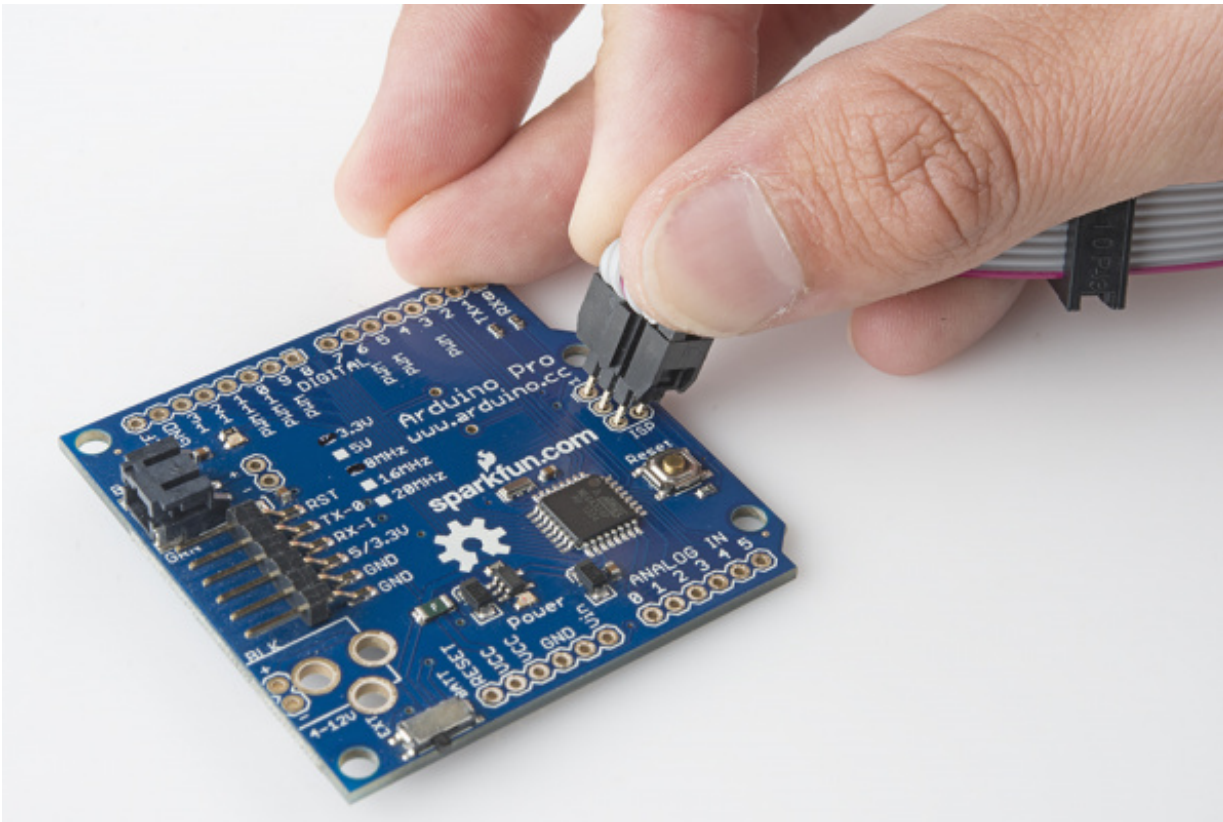
The bootloader upload process is complete when the "Burning bootloader to I/O board (this may take a minute)..." message turns to "Done burning bootloader". It really does take a while -- it's not lying when it says it "may take a minute."

Pogo Pins or the Angled Header Press

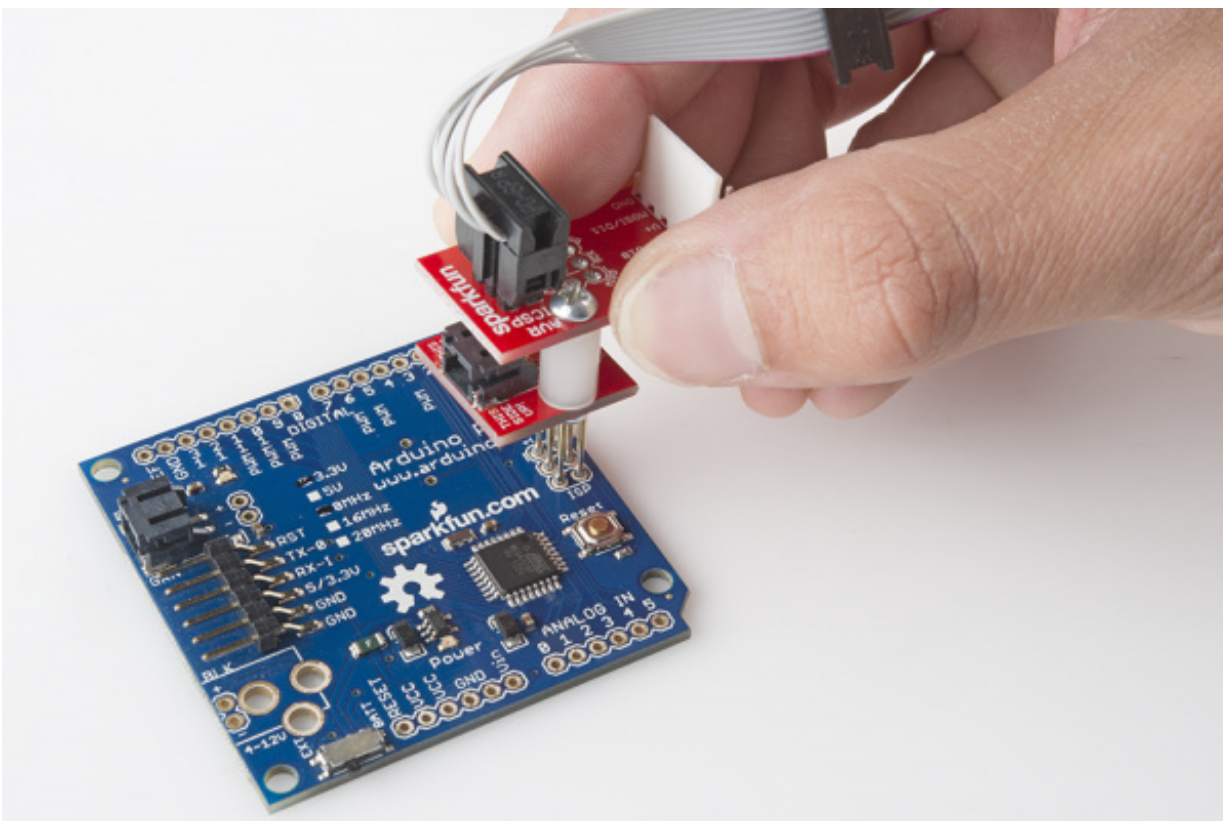
Most Arduino boards should have male pins populated on this 2x3 connector. If your board doesn't have pins shooting out of those holes, there are a few options.

You can [solder](#) a couple strips of 3 [straight male headers](#) in there, to get the best, most reliable connection. But if you want to avoid soldering, you can use those same headers ([long headers](#) work better for this), plugging the long end into the programming cable and pushing the short end into the

empty holes, while angling them to make contact on all six pins.



Another solder-less option is to use the [ISP Pogo Adapter](#), which will afford you a more reliable electrical connection.



Both of these methods can be tricky -- you have to hold those pins steady while the code uploads to your Arduino -- but they're a good solderless, temporary option.

Using AVRDUDE via Command Line

If you're looking for more control over your AVR Pocket Programmer -- and the AVR it's connected to -- follow along below. We'll demonstrate how to use [AVRDUDE](#), an open-source command line wonder-utility for reading, writing, and manipulating AVR's.

If you have Arduino, then you already have AVRDUDE installed -- it's the tool Arduino uses under the hood to upload sketches. If you need to install AVRDUDE separately, check out the [documentation under AVRDUDE's downloads](#). The **avrdude-doc-X.XX.pdf**'s (i.e. **avrdude-doc-6.3.pdf**) files are particularly useful when installing AVRDUDE for your operating system if you have issues using AVRDUDE commands in any directory via command line.

[AVRDUDE: Downloads](#)

Sanity Check -- AVRDUDE Paths

AVRDUDE is a **command-line tool**, so, in order to use it, you'll need to open up the ["Command Prompt" \(Windows\) or "Terminal" \(Mac/Linux\)](#). To make sure AVRDUDE is working it's good to do a little sanity check first. Open up the command line and type the following command.

```
language:bash
avrdude
```

You should see an output similar to the image below.


```
Command Prompt
S:\>avrdude
Usage: avrdude [options]
Options:
  -p <partno>           Required. Specify AVR device.
  -b <baudrate>         Override RS-232 baud rate.
  -B <bitclock>         Specify JTAG/STK500v2 bit clock period (us).
  -C <config-file>      Specify location of configuration file.
  -c <programmer>       Specify programmer type.
  -D                    Disable auto erase for flash memory
  -i <delay>            ISP Clock Delay [in microseconds]
  -P <port>             Specify connection port.
  -F                    Override invalid signature check.
  -e                    Perform a chip erase.
  -O                    Perform RC oscillator calibration (see AVR053).
  -U <memtype>:r|w|v:<filename>[:format]
                        Memory operation specification.
                        Multiple -U options are allowed, each request
                        is performed in the order specified.
  -n                    Do not write anything to the device.
  -V                    Do not verify.
  -u                    Disable safemode, default when running from a scri
pt.
  -s                    Silent safemode operation, will not ask you if
                        fuses should be changed back.
  -t                    Enter terminal mode.
  -E <exitspec>[,<exitspec>] List programmer exit specifications.
  -x <extended_param>   Pass <extended_param> to programmer.
  -y                    Count # erase cycles in EEPROM.
  -Y <number>           Initialize erase cycle # in EEPROM.
  -v                    Verbose output. -v -v for more.
  -q                    Quell progress output. -q -q for less.
  -?                    Display this usage.

avrdude version 5.10, URL: <http://savannah.nongnu.org/projects/avrdude/>
S:\>_
```

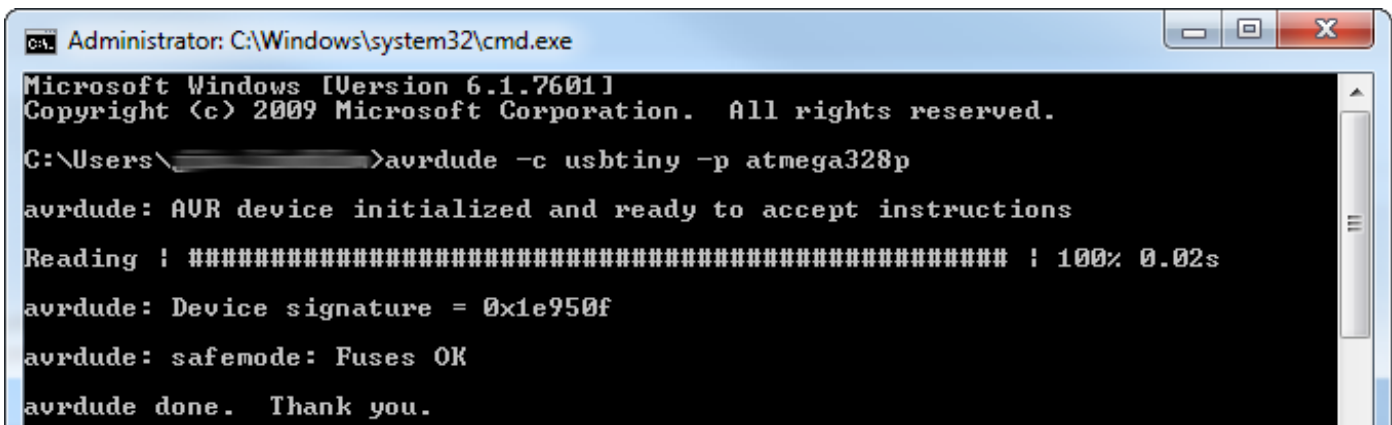
Sanity Check -- Device Signature Verification

Note: The following example is all assuming you have an **ATmega328P** connected at the other end of your programmer. If you have a different type of microcontroller, you'll need to formulate a slightly different command, check the [Specify Programmer and AVR Device section](#) below.

To make sure AVRDUDE is working, and your AVR Pocket Programmer is connected correctly, it's good to do another little sanity check. Type this into your command prompt:

```
language:bash
avrdude -c usbtiny -p atmega328p
```

If everything is connected correctly, you should get a response like this:

A screenshot of a Windows command prompt window titled "Administrator: C:\Windows\system32\cmd.exe". The window shows the output of the AVRDUDE command. The text is as follows:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\>avrdude -c usbtiny -p atmega328p

avrdude: AVR device initialized and ready to accept instructions

Reading : ##### : 100% 0.02s

avrdude: Device signature = 0x1e950f

avrdude: safemode: Fuses OK

avrdude done. Thank you.
```

This basic command defines the programmer type you're using and the AVR it's talking to. AVRDUDE will attempt to read the **Device Signature** from your AVR, which is different for each AVR type out there. Every **ATmega328P** should have a device signature of 0x1E950F.

Flash Programming

Heads up! This example assumes that the fuse bits (i.e. the low, high, and extended) are set already before flashing the .hex file. Since we are using a RedBoard Programmed with Arduino, we can flash a .hex file to the board.

Now that you've verified that everything is in working order, you can do all sorts of memory reading and writing with AVRDUDE. The main piece of memory you probably want to write is flash -- the non-volatile memory where the programs are stored.

Warning! By writing the following **blink.hex** file to your AVR microcontroller, this will overwrite what is in memory. The file does not have an Arduino bootloader so you will not be able to upload via serial using the Arduino IDE until you reinstall the bootloader with your respective microcontroller. If you are following along with the RedBoard Programmed with Arduino, you can still reinstall the bootloader to upload via serial again. The board uses the **optiboot_atmega328.hex**. You can find this in the Arduino program folder similar to this path ...**\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot** or in "**Reinstalling the RedBoard's Arduino Bootloader**" later in this tutorial.

This example will be using the **blink.hex** file as an example. Download the following files below. If you are using the **blink.hex** file, make sure that you unzip the folder and place it in the working directory.

[Download Blink Here \(ZIP\)](#)

This command will perform a basic write to flash using the HEX file.

```
language:bash
avrdude -c usbtiny -p atmega328p -U flash:w:blink.hex
```

Writing to flash will take a little longer than reading the signature bits. You'll see a text status bar scroll by as the device is read, written to, and verified.


```
Administrator: C:\Windows\system32\cmd.exe
C:\>cd blink
C:\blink>avrdude -c usbtiny -p atmega328p -U flash:w:blink.hex
avrdude: AVR device initialized and ready to accept instructions
Reading : ##### : 100% 0.02s
avrdude: Device signature = 0x1e950f
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
        To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "blink.hex"
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: writing flash (3098 bytes):
Writing : ##### : 100% 9.67s
avrdude: 3098 bytes of flash written
avrdude: verifying flash memory against blink.hex:
avrdude: load data flash data from input file blink.hex:
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: input file blink.hex contains 3098 bytes
avrdude: reading on-chip flash data:
Reading : ##### : 100% 5.51s
avrdude: verifying ...
avrdude: 3098 bytes of flash verified
avrdude: safemode: Fuses OK
avrdude done. Thank you.
```

The -U option command handles all of the memory reads and writes. We tell it we want to work with flash memory, do a write with w, and then tell it the location of the hex file we want to write.

Tip: In some cases, you may need to specify the *.hex file that you are flashing. This is usually optional but you may get an error if AVRDUDE is not able to read the file.

Invalid File Format

To specify, you can by add an :i to indicate that it is an Intel hex format:

```
avrdude -c usbtiny -p atmega328p -U flash:w:blink.hex:i
```

Or :a to auto detect the format:

```
avrdude -c usbtiny -p atmega328p -U flash:w:blink.hex:a
```

For more information, try checking the AVRDUDE's Online Documentation under the Option Description where it describes the command "-U memtype:op:filename[:format]".

[AVRDUDE Online Documentation: Option Descriptions](#)

Flash Reading

The -U command can also be used to read the memory contents of an AVR. A command like below, for example, will read the contents of your AVR and store them into a file called "**mystery.hex**".

```
language:bash
avrdude -c usbtiny -p atmega328p -U flash:r:mystery.hex:r
```

This is incredibly useful if you want to copy the contents of one Arduino to another. Or maybe you're a masochist, and you want to try reverse-engineering the mystery code in an AVR.

Reinstalling the RedBoard's Arduino Bootloader

Now that you have a hang of flashing hex files to your RedBoard, try reinstalling the bootloader with the following file. Download the file.

[RedBoard Programmed with Arduino Bootloader \(HEX\)](#)

Navigate to the path where you downloaded the bootloader and enter the following command.

```
language:bash
avrdude -c usbtiny -p atmega328p -U flash:w:optiboot_atmega328_2012_with_1s_watchdog.hex
```

If all goes well, you should get a message indicating that it was written, verified, and finished uploading. You should get an output similar to the output below. In this case, the configuration file (i.e. **avrdude.conf**) and the bootloader (***.hex**) were not located in the same working directory. Two additional commands were needed to specify where to look for the files. Additionally, the ***.hex** format needed to be autodetected when flashing the file by adding the :a.

```
Command Prompt

C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude -C "C:\Program Files\arduino-1.8.5\hardware\tools\avr\etc\avrdude.conf" -c usbtiny -p atmega328p -U flash:w:"C:\Program Files\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot\optiboot_atmega328.hex":a

avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.02s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
        To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "C:\Program Files\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot\optiboot_atmega328.hex"
avrdude: input file C:\Program Files\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot\optiboot_atmega328.hex auto detected as Intel Hex
avrdude: writing flash (32768 bytes):

Writing | ##### | 100% 0.02s

avrdude: 32768 bytes of flash written
avrdude: verifying flash memory against C:\Program Files\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot\optiboot_atmega328.hex:
avrdude: load data flash data from input file C:\Program Files\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot\optiboot_atmega328.hex:
avrdude: input file C:\Program Files\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot\optiboot_atmega328.hex auto detected as Intel Hex
avrdude: input file C:\Program Files\arduino-1.8.5\hardware\arduino\avr\bootloaders\optiboot\optiboot_atmega328.hex contains 32768 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 0.01s

avrdude: verifying ...
avrdude: 32768 bytes of flash verified

avrdude: safemode: Fuses OK (E:FD, H:DE, L:FF)

avrdude done. Thank you.

C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>
```

Head back to the Arduino IDE, disconnect the USB cable from your programmer, and connect to the RedBoard's USB port to see if you can upload a simple **blink.ino** sketch back to the board via serial. Make sure to select the appropriate board definition and COM port before uploading. You should see a familiar message indicating that the Arduino IDE is *"Done Uploading"* when complete and the on-board LED begin to blink on pin 13.

Note: Looking for more information about [Installing an Arduino Bootloader](#)? Check out our tutorial below about the different methods of flashing an Arduino bootloader to an AVR chip!

[Installing an Arduino Bootloader](#)

[December 4, 2013](#)

This tutorial will teach you what a bootloader is and why you would need to install or reinstall it. We will also go over the process of burning a bootloader by flashing a hex file to an Arduino

microcontroller.

[Favorited Favorite](#) 25

Useful Options

Here are just a few last AVRDUDE tips and tricks before we turn you loose on the AVR world.

[Specify Programmer and AVR Device](#)

Two options required for using AVRDUDE are the **programmer type** and **AVR device** specification:

- The **programmer** definition, assuming you're using the AVR Pocket Programmer, will be `-c usbtiny`. If you need to use a different programmer [check out this page](#) and CTRL+F to "**-c programmer-id**".
- The **AVR device** type is defined with the `-p` option. We've shown a few examples with the ATmega328P, but what if you're using an ATtiny85? In that case, you'll want to put `-p t85` instead. Check out the [top of this page](#) for an exhaustive list of compatible AVR device types.

Verbose Output

Adding one, or more `-v`'s to your AVRDUDE command will enable various levels of verbosity to the action. This is handy if you need a summary of your configuration options, or an in-depth view into what data is being sent to your AVR.

There's plenty more where that came from. Check out the [AVRDUDE Online Documentation under Option Descriptions](#) for the entire list of commands.

[AVRDUDE Online Documentation: Option Descriptions](#)

Troubleshooting

Below are a few troubleshooting tips for resolving some of the AVRDUDE errors that you may run into.

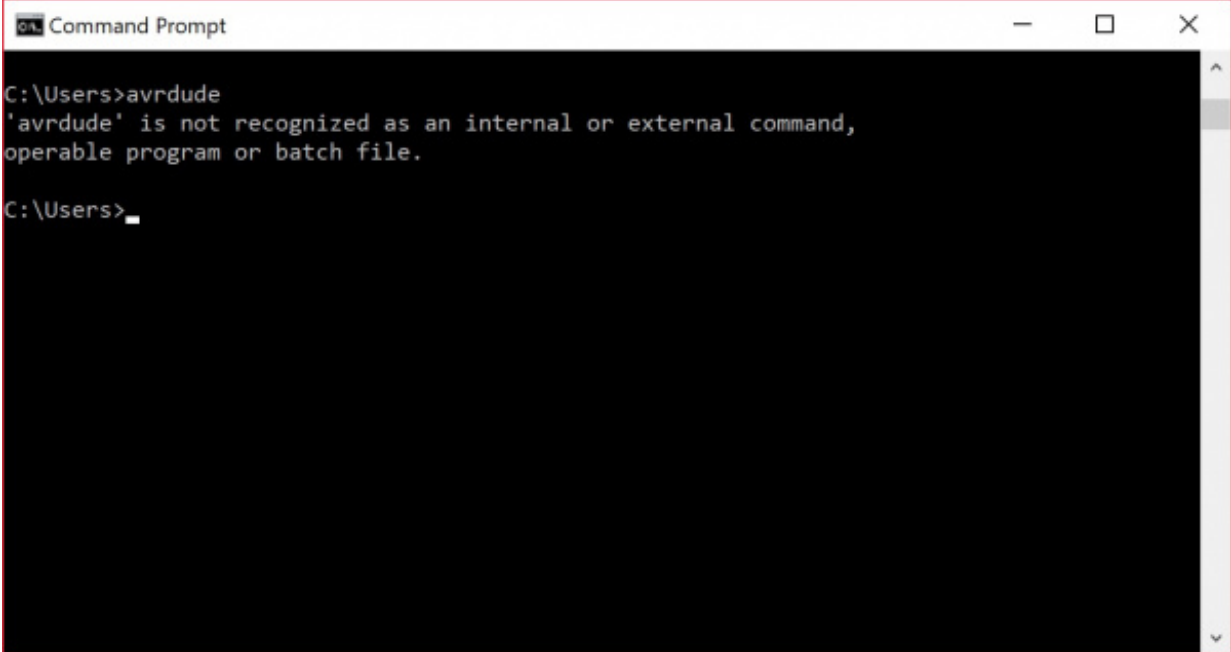
AVRDUDE Not Recognized

If you are having issues getting a response from AVRDUDE, you may receive the following error. It's probably due to certain environmental variables or your computer settings preventing you from properly using AVRDUDE.

```
language:bash
```

'avrdude' is not recognized as an internal or external command, operable program or batch file

The error output in the command line may look similar to the screenshot below.

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The command prompt shows the user at the C:\Users directory. They entered the command 'avrdude'. The output is an error message: "'avrdude' is not recognized as an internal or external command, operable program or batch file." The prompt then returns to C:\Users>.

```
Command Prompt
C:\Users>avrdude
'avrdude' is not recognized as an internal or external command,
operable program or batch file.
C:\Users>
```

One solution may be to try following the instructions provided by AVRDUDE to install it for your OS. For Windows, you could automatically install **WinAVR 20100110** as explained briefly on page 35 of the AVRDUDE documents v6.3.

[AVRDUDE Documents v6.3 \(PDF\)](#)

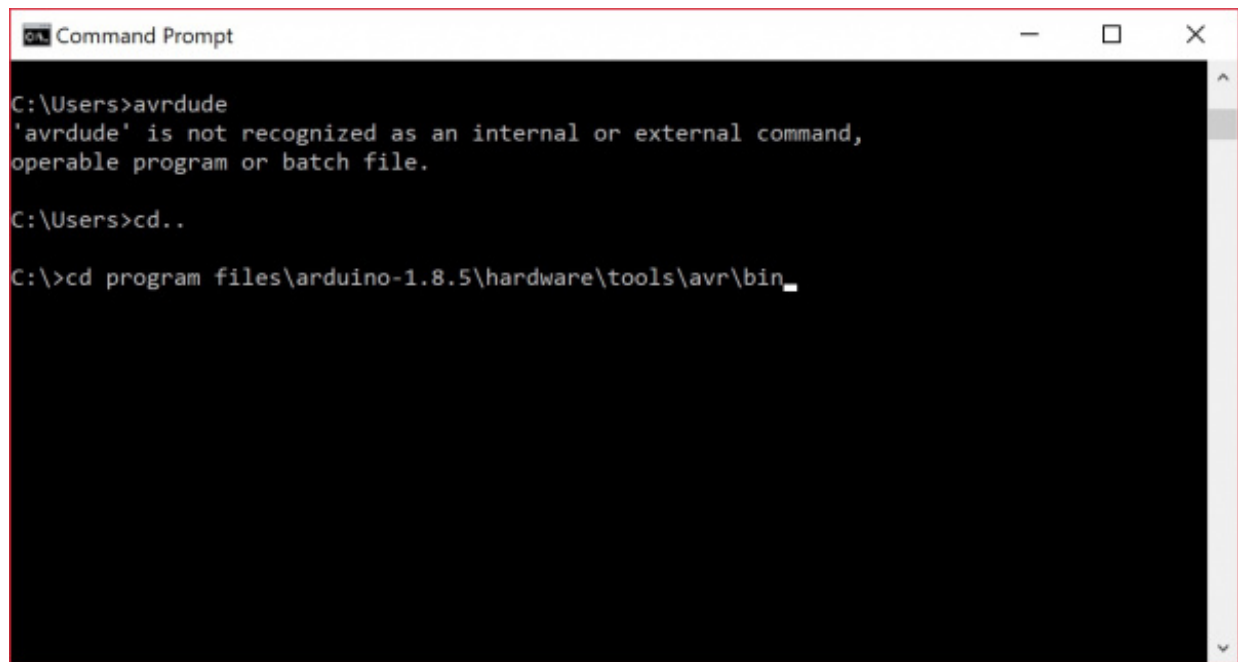
Otherwise, you can move to the Arduino IDE program folder where **avrdude.exe** is located. Try doing a search within the Arduino program folder to determine the path. Then navigate to the location where it is located using the **cd..** and **cd** commands. In this case, Arduino IDE v1.8.5 was installed and located in the Program Files folder of my **C:** drive under **...program files\arduino-1.8.5\hardware\tools\avr\bin**. Type in the change directory commands to navigate to the proper location in the command line. From the screenshot of the error, I needed to move up the directory by using the following command.

```
language:bash
cd..
```

Then I needed to move into the Arduino's program folder that is located in the **C:** drive.

```
language:bash
cd programfiles\arduino1.8.5\hardware\tools\avr\bin
```

Your command line should look similar to the image below.



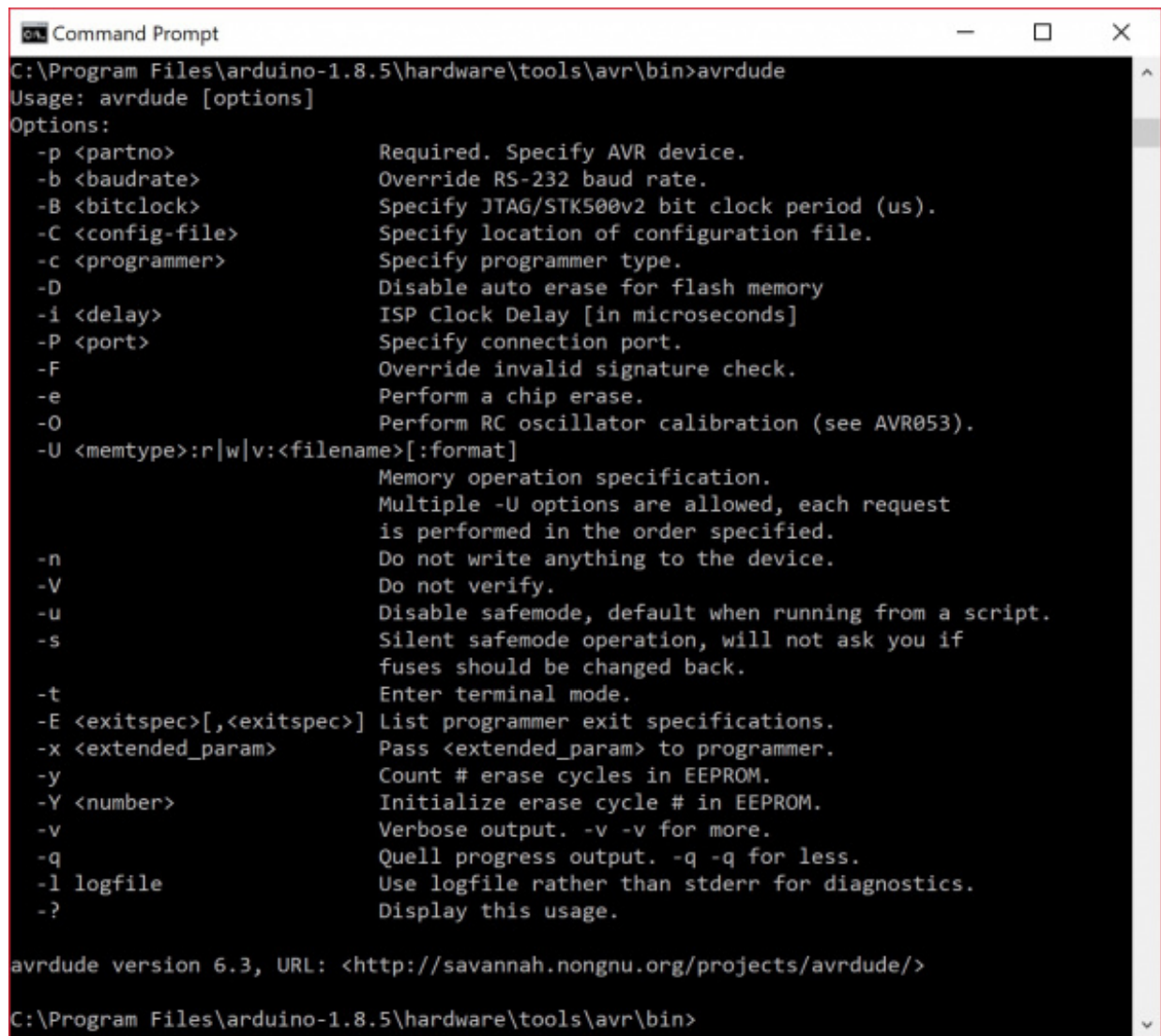
```
Command Prompt

C:\Users>avrdude
'avrdude' is not recognized as an internal or external command,
operable program or batch file.

C:\Users>cd..

C:\>cd program files\arduino-1.8.5\hardware\tools\avr\bin_
```

Once you are in the proper working directory, type in `avrdude` again. You should see an output similar to the image below.

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the command `C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude` and its output. The output displays the usage and options for the `avrdude` command. The options listed include `-p` for specifying the AVR device, `-b` for baud rate, `-B` for bit clock period, `-C` for configuration file, `-c` for programmer type, `-D` for disabling auto erase, `-i` for ISP clock delay, `-P` for connection port, `-F` for overriding signature check, `-e` for chip erase, `-O` for RC oscillator calibration, `-U` for memory operation specification, `-n` for no write, `-V` for no verify, `-u` for disabling safemode, `-s` for silent safemode, `-t` for terminal mode, `-E` for exit specifications, `-x` for extended parameters, `-y` for erasing EEPROM, `-Y` for initializing EEPROM, `-v` for verbose output, `-q` for quiet output, `-l` for logfile, and `-?` for displaying usage. The version of `avrdude` is 6.3, and the URL is <http://savannah.nongnu.org/projects/avrdude/>. The prompt returns to `C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>`.

```
Command Prompt
C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude
Usage: avrdude [options]
Options:
  -p <partno>           Required. Specify AVR device.
  -b <baudrate>         Override RS-232 baud rate.
  -B <bitclock>         Specify JTAG/STK500v2 bit clock period (us).
  -C <config-file>      Specify location of configuration file.
  -c <programmer>       Specify programmer type.
  -D                    Disable auto erase for flash memory
  -i <delay>            ISP Clock Delay [in microseconds]
  -P <port>             Specify connection port.
  -F                    Override invalid signature check.
  -e                    Perform a chip erase.
  -O                    Perform RC oscillator calibration (see AVR053).
  -U <memtype>:r|w|v:<filename>[:format]
                        Memory operation specification.
                        Multiple -U options are allowed, each request
                        is performed in the order specified.
  -n                    Do not write anything to the device.
  -V                    Do not verify.
  -u                    Disable safemode, default when running from a script.
  -s                    Silent safemode operation, will not ask you if
                        fuses should be changed back.
  -t                    Enter terminal mode.
  -E <exitspec>[,<exitspec>] List programmer exit specifications.
  -x <extended_param>   Pass <extended_param> to programmer.
  -y                    Count # erase cycles in EEPROM.
  -Y <number>           Initialize erase cycle # in EEPROM.
  -v                    Verbose output. -v -v for more.
  -q                    Quell progress output. -q -q for less.
  -l logfile            Use logfile rather than stderr for diagnostics.
  -?                    Display this usage.

avrdude version 6.3, URL: <http://savannah.nongnu.org/projects/avrdude/>

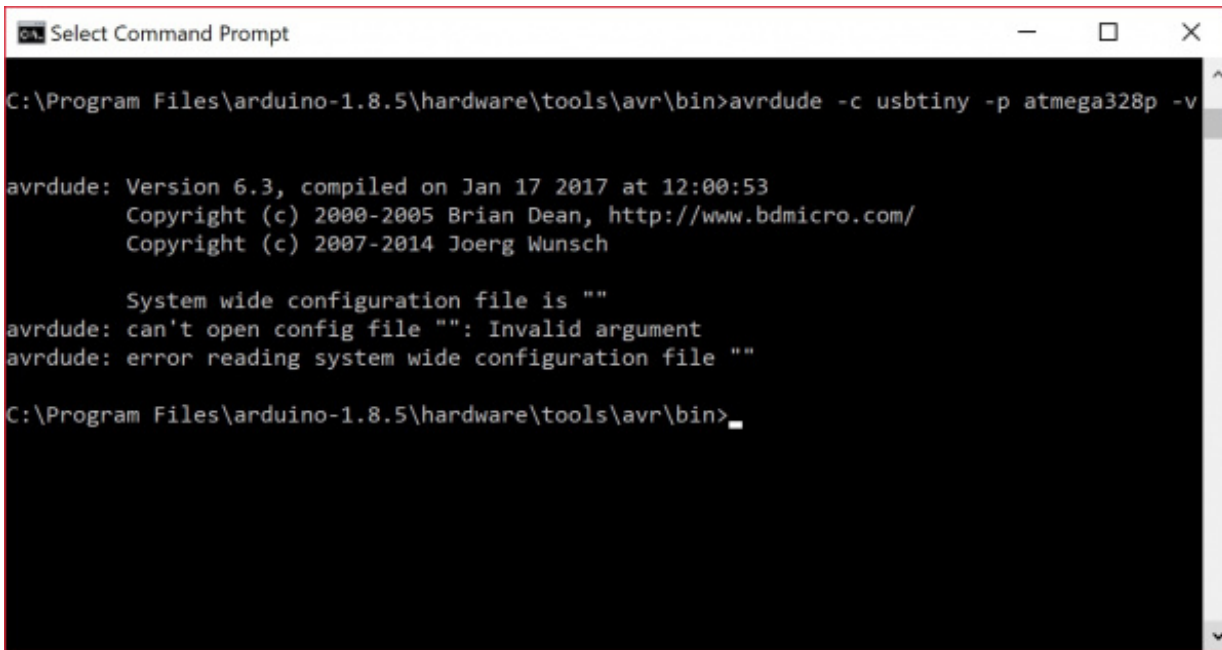
C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>
```

Configuration File Not Found

If you are having trouble reading the AVR device signature to verify the device using the command `avrdude -c usbtiny -p atmega328P`; and you receive this error:

```
language:bash
System wide configuration file is ""
avrdude: can't open config file "": Invalid argument
avrdude: error reading system wide configuration file ""
```

It's probably due to the way AVRDUDE was installed on a computer. In this case, AVRDUDE could not find the location of the **avrdude.conf** file. This is probably due to environmental variables or your computer settings preventing you from properly using AVRDUDE. If you remember from the earlier troubleshooting tip, AVRDUDE was located in the Arduino IDE program folder. While the working directory was correct, the **avrdude.conf** file was in a different folder as you can see from the image below on a Windows OS.



```
C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude -c usbtiny -p atmega328p -v

avrdude: Version 6.3, compiled on Jan 17 2017 at 12:00:53
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2014 Joerg Wunsch

System wide configuration file is ""
avrdude: can't open config file "": Invalid argument
avrdude: error reading system wide configuration file ""

C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>
```

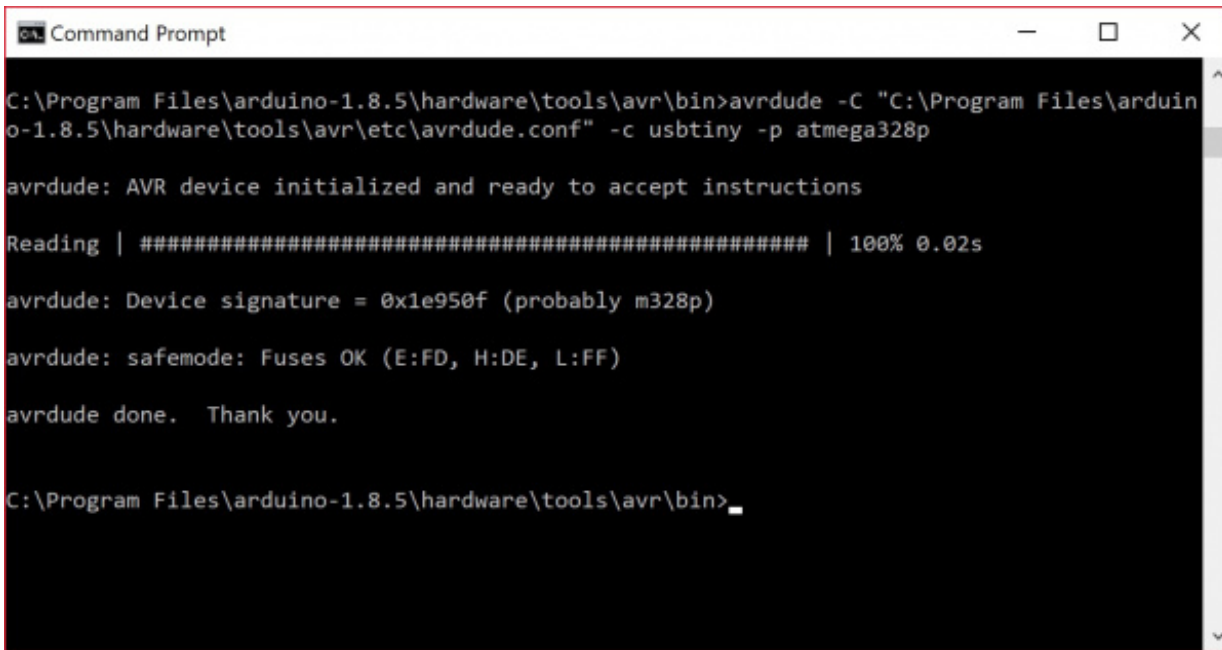
The easiest solution would be to adjust the environmental settings by automatically installing it for your OS as explained in the [avrdude-docs \(v6.3\)](#). For Windows, you could install **WinAVR 20100110** as explained briefly on page 35 of the AVRDUDE documents v6.3.

[AVRDUDE Documents v6.3 \(PDF\)](#)

Otherwise, you could use the -C command and provide the path in quotes ("...\\avrdude.conf") where the **avrdude.conf** file is located. For the Arduino IDE v1.8.5, it was located in...**arduino-1.8.5\\hardware\\tools\\avr\\etc** directory. Assuming that you have AVRDUDE in the working directory, the command should be similar to the command below to read an ATmega328P.

```
language:bash
avrdude -C "C:\Program Files\arduino-1.8.5\hardware\tools\avr\etc\avrdude.conf" -c usbtiny -p atmega328p
```

A successful device signature read with the configuration file should look similar to the output below.



```
Command Prompt
C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude -C "C:\Program Files\arduino-1.8.5\hardware\tools\avr\etc\avrdude.conf" -c usbtiny -p atmega328p

avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.02s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: safemode: Fuses OK (E:FD, H:DE, L:FF)
avrdude done. Thank you.

C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>
```

Driver Related Issues

If you run AVRDUDE commands and receive this error below, the issue may be related to the drivers for the AVR Programmer whose device ID is 0x1781/0xc9f. Either the drivers are not installed or there is a driver conflict.

```
language:bash
avrdude: Error: Could not find USBtiny device (0x1781/0xc9f)
```

Drivers Not Installed

One solution is to ensure that the [drivers are installed](#) as explained earlier. You may also want to try another USB cable or unplugging/replugging the AVR programmer back into your COM port. The error output in the command line may look similar to the screenshot below.

```
Command Prompt

C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude -C "C:\Program Files\arduino-1.8.5\hardware\tools\avr\etc\avrdude.conf" -c usbtiny -p atmega328p -v

avrdude: Version 6.3, compiled on Jan 17 2017 at 12:00:53
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2014 Joerg Wunsch

System wide configuration file is "C:\Program Files\arduino-1.8.5\hardware\tools\avr\etc\avrdude.conf"

Using Port                : usb
Using Programmer           : usbtiny
avrdude: Error: Could not find USBtiny device (0x1781/0xc9f)

avrdude done. Thank you.

C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>
```

Driver Conflicts

If you have installed the correct drivers as explained earlier, it's possible that there is a driver conflict. You'll receive the same error but the solution may not be as intuitive as you may think. The output in the screenshot below occurred when using the Tiny AVR Programmer to verify an ATtiny85's device signature. The drivers were installed correctly and had been working with the Pocket AVR Programmer. However, the Tiny AVR Programmer was still not recognized.

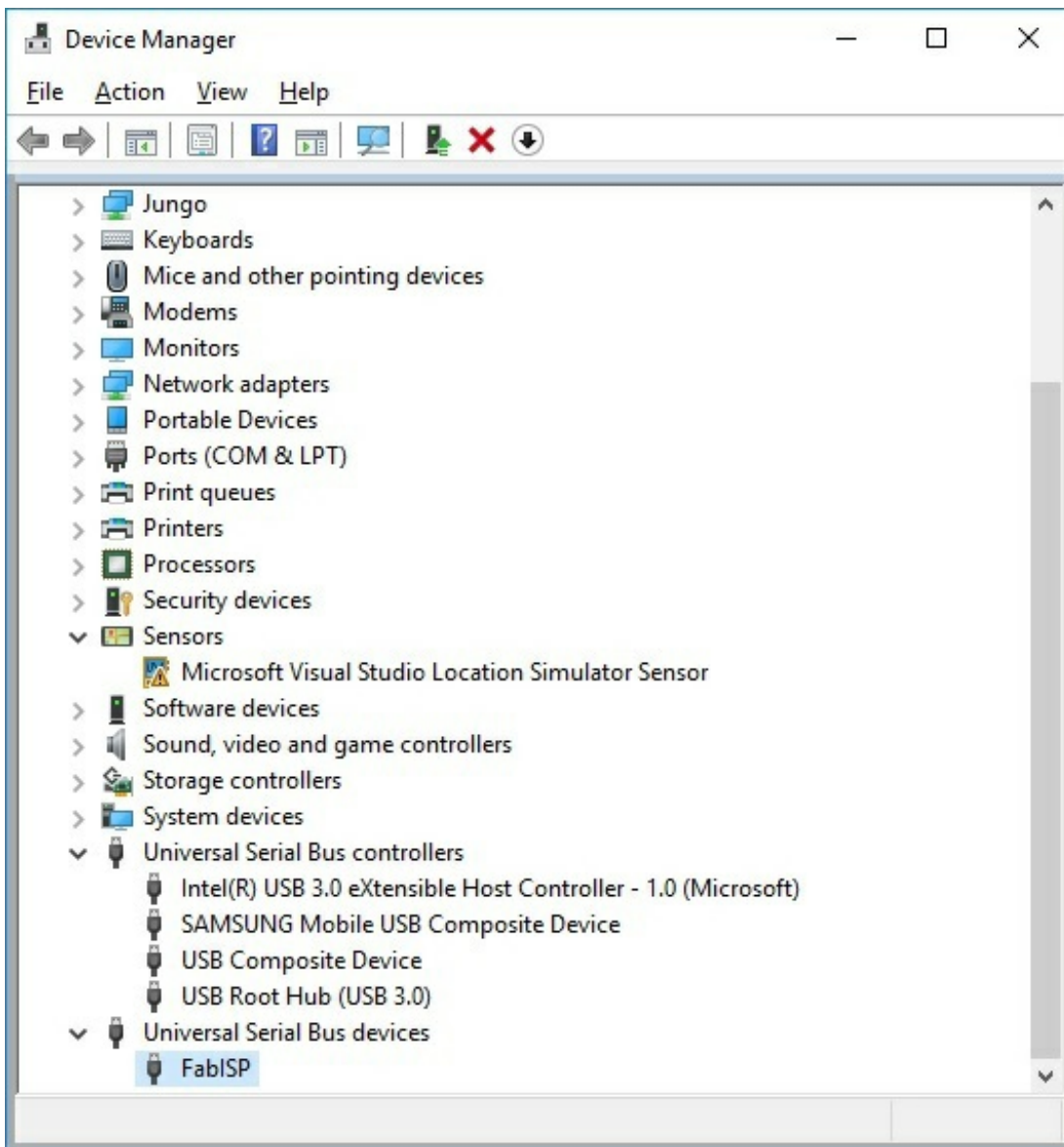
```
Command Prompt

S:\>avrdude -c usbtiny -p t85
avrdude: Error: Could not find USBtiny device (0x1781/0xc9f)

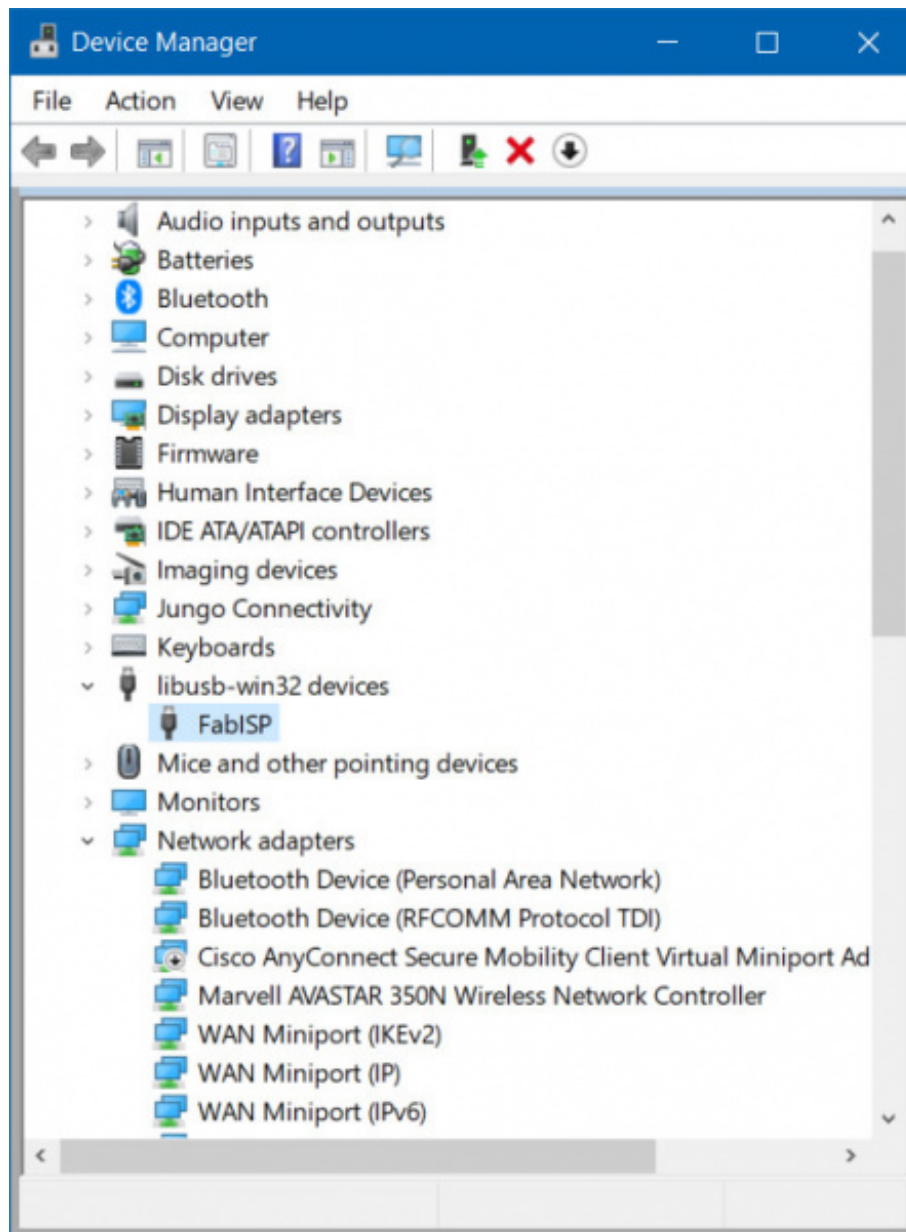
avrdude done. Thank you.

S:\>
```

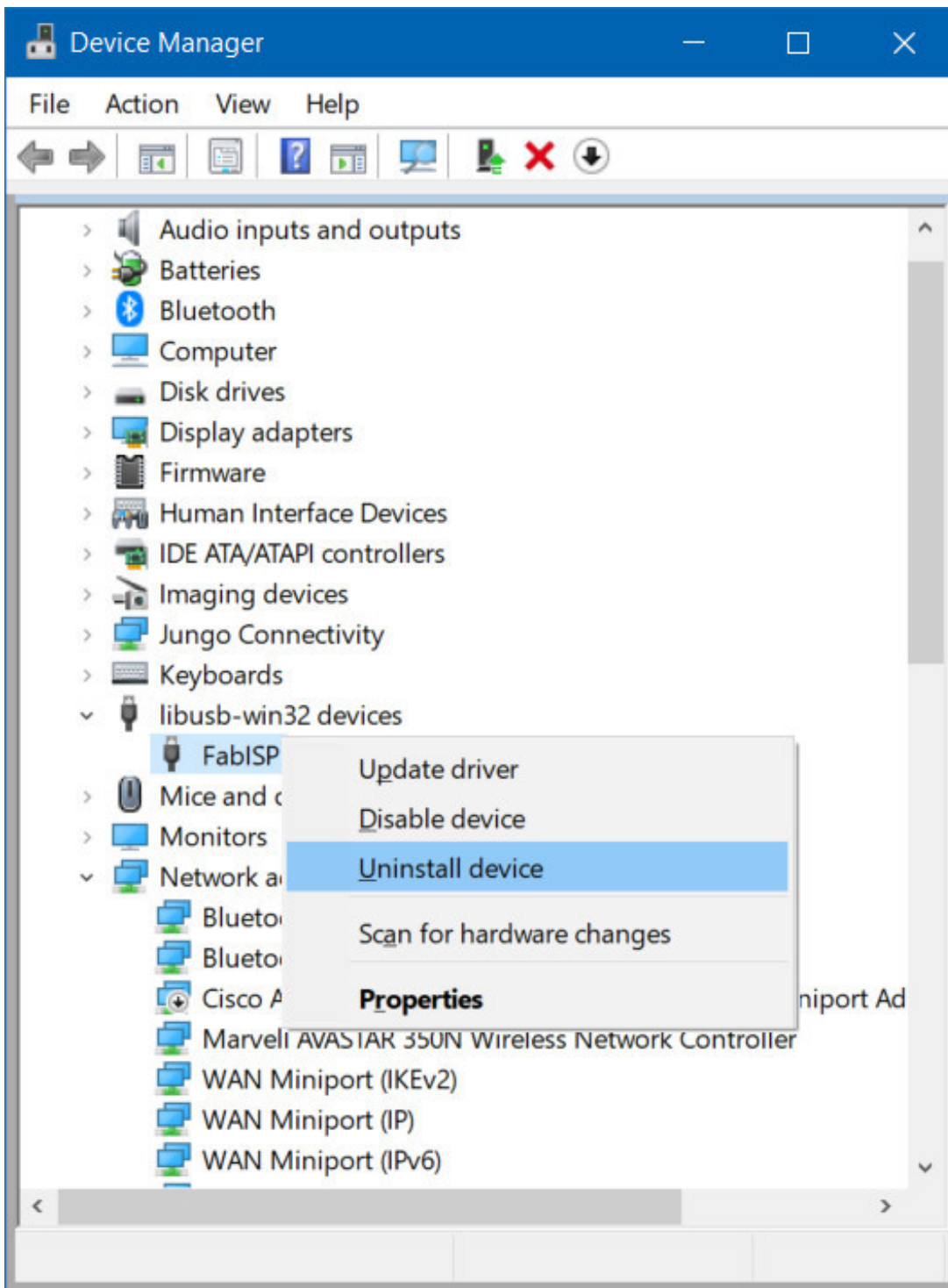
Opening up the device manager, the Tiny AVR Programmer showed up as a different driver (i.e. **FabISP**) and name as shown below.



Note: In some cases, the driver may appear under **libusb-win32 devices > FabISP**. If the drivers listed under the **libusb-win32 devices** tree fail to work, you will need to follow the directions to reinstall the driver.



The solution was to right click and delete the driver. Simply right click the COM port that it enumerated on and select "**Uninstall device**".



You may see a window pop up similar to the image below. Click on the button labeled **Uninstall**. In some cases, Windows may provide an option to **"Delete the driver software for this device."** if the option is provided, simply mark the checkbox before clicking on the button to uninstall.



After uninstalling, power cycle the programmer by unplugging/replugging the Tiny AVR Programmer from the USB port. Head back to the Installing Drivers section and follow the instructions to [Automatically Install the Drivers using Zadig](#).

[Installing Drivers: Automatically Install the Drivers with Zadig](#)

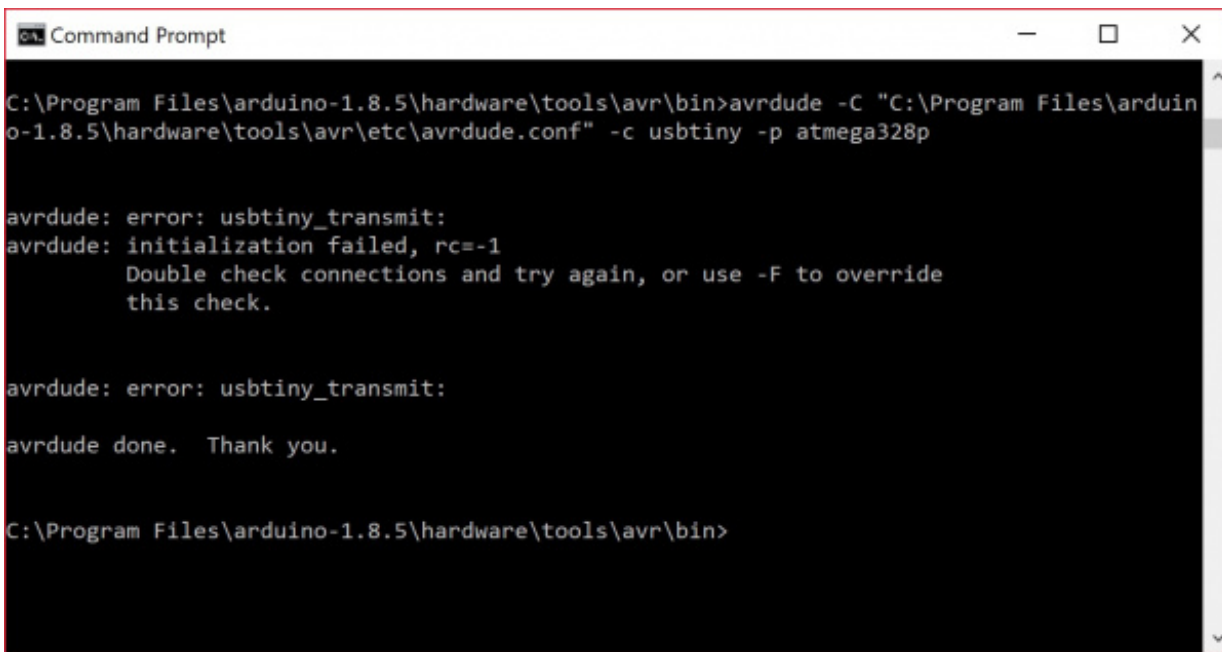
Error Connecting To AVR Programmer

If you receive an error similar to the output below, it is probably due to the connection to the AVR programmer.

```
language:bash
avrdude: error: usbtiny_transmit:
avrdude: initialization failed, rc=-1
    Double check connections and try again, or use -F to override this check.
```

```
avrdude:error: usbtiny_transmit:
```

One solution is to try to unplug and replug the AVR programmer back to your COM port. You may also want to check the USB cable or ensure that the drivers are installed correctly. The output in the command line may look similar to the screenshot below.

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The command entered is: `C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude -C "C:\Program Files\arduino-1.8.5\hardware\tools\avr\etc\avrdude.conf" -c usbtiny -p atmega328p`. The output shows two "error: usbtiny_transmit:" messages, followed by "avrdude: initialization failed, rc=-1" and a suggestion to "Double check connections and try again, or use -F to override this check." The command then says "avrdude done. Thank you." and the prompt returns to `C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>`.

```
Command Prompt
C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>avrdude -C "C:\Program Files\arduino-1.8.5\hardware\tools\avr\etc\avrdude.conf" -c usbtiny -p atmega328p

avrdude: error: usbtiny_transmit:
avrdude: initialization failed, rc=-1
        Double check connections and try again, or use -F to override
        this check.

avrdude: error: usbtiny_transmit:
avrdude done.  Thank you.

C:\Program Files\arduino-1.8.5\hardware\tools\avr\bin>
```

Heads Up! The following are only a few errors that we have run into when using AVRDUDE. To browse other common issues and how to troubleshoot, you may want to try checking the AVRDUDE online manual.

[AVRDUDE Online Manual: Troubleshooting](#)

Or do a search online with the error that you are having and check different forums to see if anyone else has run into the same issues as you have.

Resources and Going Further

Now that you've successfully got your Pocket AVR Programmer up and running, it's time to incorporate it into your own project!

Here are some more AVR Pocket Programmer related resources, should you need them:

- [Schematic \(PDF\)](#)
- [Eagle Files \(ZIP\)](#)
- GitHub
 - [Product Repo](#) -- Here you'll find everything from PCB design files, and firmware to custom enclosure designs.
 - [Firmware](#)
- Drivers
 - [Zadig v2.0.1.160 Software and USBtiny \(ZIP\)](#) -- For automatic installation
 - [USBTiny libusb-win32 \(ZIP\)](#) -- For manual installation
 - [GitHub: Signed USBTiny Drivers](#) -- If the first 3 options fail to install
- [AVRDUDE Online Manual](#)
 - [AVRDUDE Troubleshooting](#) -- Common errors can be found in this troubleshooting section.

We've got plenty more tutorials where that came from. If you're looking for more stuff to learn, or are looking for some project inspiration, check out these tutorials!

- [Installing an Arduino Bootloader](#) -- Use your AVR Pocket Programmer to load a bootloader onto an Arduino. This tutorial will teach you what a bootloader is, why you would need to install/reinstall it, and go over the process of doing so.
- [Tiny AVR Programmer Hookup Guide](#) -- If you're looking to program ATtiny85's specifically, check out the [Tiny AVR Programmer](#).
- [Using the Arduino Pro Mini 3.3V](#) -- If you're already directly programming your Arduino, take it a step further with the Arduino Pro Mini.
- [Wireless Arduino Programming with Electric Imp](#) -- If you're feeling constrained by the USB cables, check out this tutorial where we upload code to an Arduino wirelessly!
- [Wireless XBee/AVR Bootloading](#) -- Use your AVR Pocket Programmer to upload a custom bootloader, then wirelessly program your Arduino.

[**Installing an Arduino Bootloader**](#)

This tutorial will teach you what a bootloader is and why you would need to install or reinstall it. We will also go over the process of burning a bootloader by flashing a hex file to an Arduino microcontroller.

[Favorited Favorite](#) 25

[**Using the Arduino Pro Mini 3.3V**](#)

This tutorial is your guide to all things Arduino Pro Mini. It explains what it is, what it's not, and how to get started using it.

[Favorited Favorite](#) 14

[**Tiny AVR Programmer Hookup Guide**](#)

A how-to on the Tiny AVR Programmer. How to install drivers, hook it up, and program your favorite Tiny AVR's using AVRDUDE!

[Favorited Favorite](#) 10

[**Wireless Arduino Programming with Electric Imp**](#)

Reprogram your Arduino from anywhere in the world using the Tomatoless Boots wireless bootloader with the Electric Imp.

[Favorited Favorite](#) 9

Are you looking to use a Pi to flash larger file sizes to your AVR microcontrollers? Try checking out the Pi AVR Programmer Hat!

[**Pi AVR Programmer HAT Hookup Guide**](#)

[**July 26, 2018**](#)

In this tutorial, we will use a Raspberry Pi 3 and the Pi AVR Programmer HAT to program an ATmega328P target. We are going to first program the Arduino bootloader over SPI, and then upload an Arduino sketch over a USB serial COM port.

[Favorited Favorite](#) 2

learn.sparkfun.com | [CC BY-SA 3.0](#) | SparkFun Electronics | Niwot, Colorado