sparkfun

Pocket AVR Programmer Hookup Guide
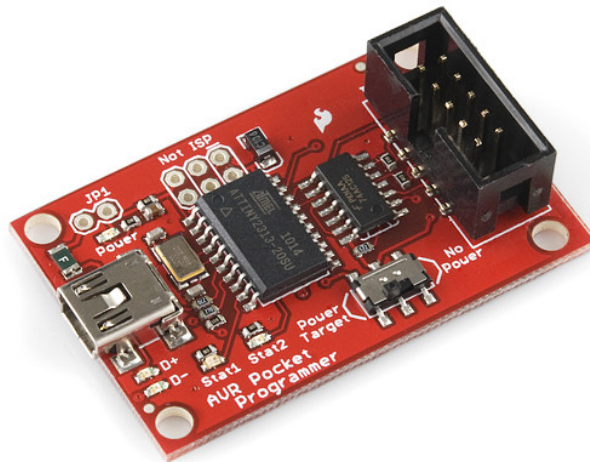
CONTRIBUTORS: 🧑 *JIMB0*

♡ FAVORITE   0

Introduction

Do you need more control over your AVRs? Whether it's an ATmega328, ATmega32U4, ATtiny85, ATmega128RFA1, if it's an AVR there's a good chance the AVR Pocket Programmer can program it.

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 – A more advanced, command-line-based approach to using the AVR Pocket Programmer.

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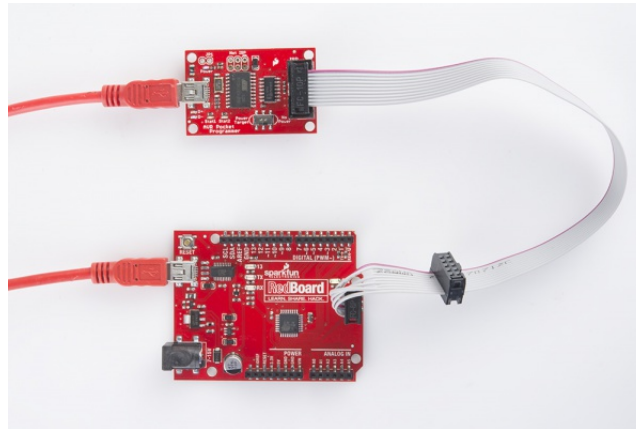
That microcontroller-to-be-programmed can be any AVR with **64K or less of flash**. The ATmega328 on an Arduino Uno or RedBoard works perfectly, but the ATmega2560 of an Arduino Mega *does not*.

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.
- AVR Programming Adapter – If you AVR is living on a breadboard, you probably don't have an interface to the standard 2x3 ISP pinout. This simple breakout board makes interfacing the programmer with your breadboarded circuit possible.

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".

Board Overview

Before we get to using the AVR Pocket Programmer, let's quickly overview what components fill the board out:
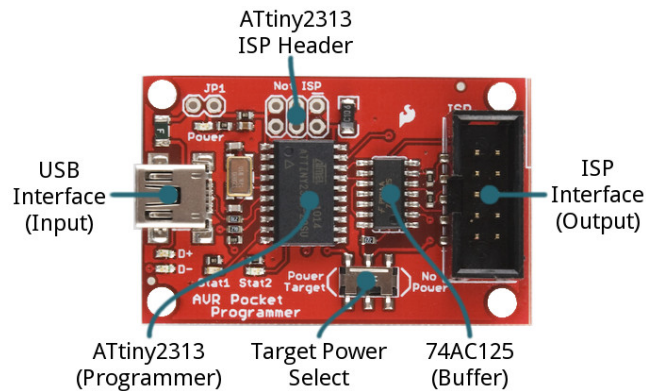
- **USB Connector** – This is your **data and power input** to the Programmer. A mini-B USB cable plugs in here and connects your computer to the Programmer.
- **2x5 ISP Header** – This shrouded header mates with the included Programming Cable, and allows you to send the programming signals out to your 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 6-pin, 2x3 connector and another 10-pin, 2x5:


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.
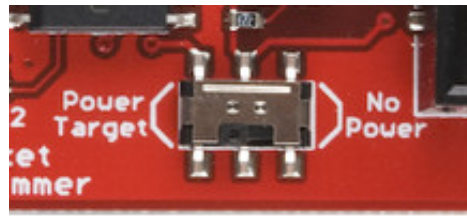
If the switch is in the *Power Target* position, it will route 5V out to your AVR. Otherwise, if the switch is pointing towards *No Power*, no signal will be connected to the 5V pin on the ISP connector.

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.

Installing Drivers

Driver installation is required on **Windows machines only**. If you're using Mac or Linux, feel free to click over to the next section. Otherwise, follow along below as we overview the installation process.

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.

Install the Drivers Automatically 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 drivers**:

Download the Zadig USBtiny Drivers

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

After you've plugged the AVR Pocket 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**.

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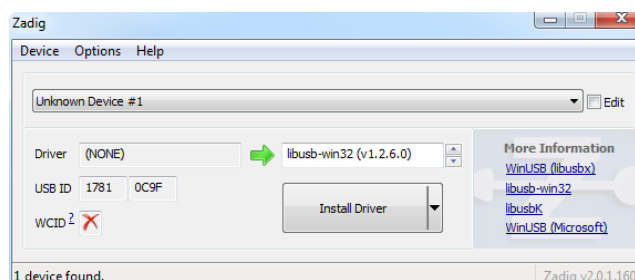
There are a few options to verify before installing the driver:

- **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.

If you were successful, close out of the Zadig program and proceed to the next section!

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.

Note: If you are using a **Windows 8** machine, before you can install the drivers you'll need to **disable driver signature enforcement**. Follow along with our tutorial to turn that overzealous safety guard off for a minute.

Click the link below to **download the drivers**:
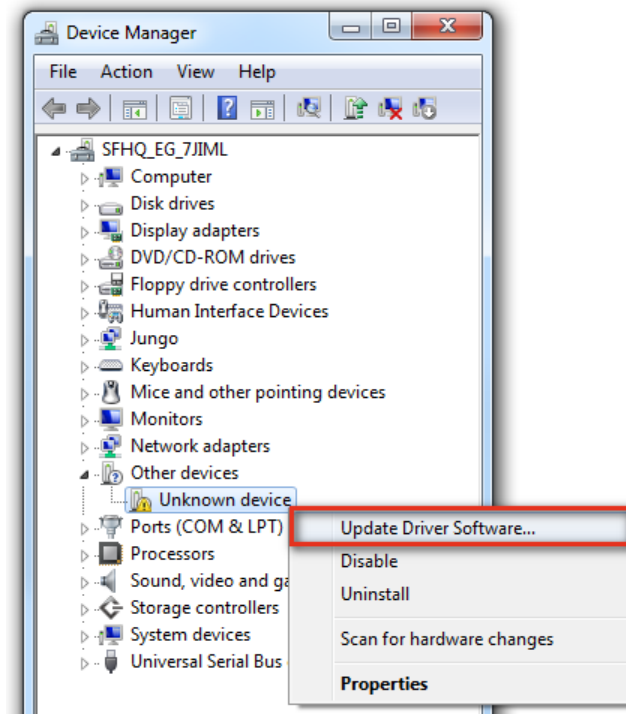
<p align="center">Download the USBtiny Drivers</p>

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**.
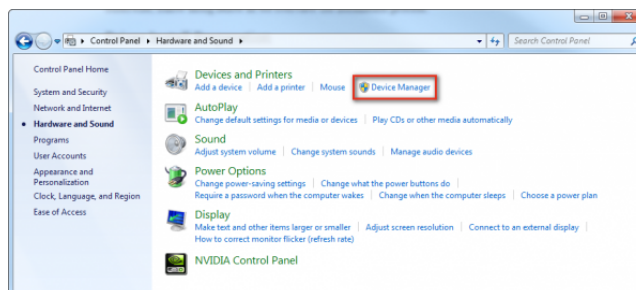

    - Or, simply open the **run tool** (press Windows Key + R), and run `devmgmt.msc`.
2. In the Device Manager, you should see an "Unknown device" listed under the "Other devices" tree. **Right click "Unkown Device"** and select **Update driver software…**.

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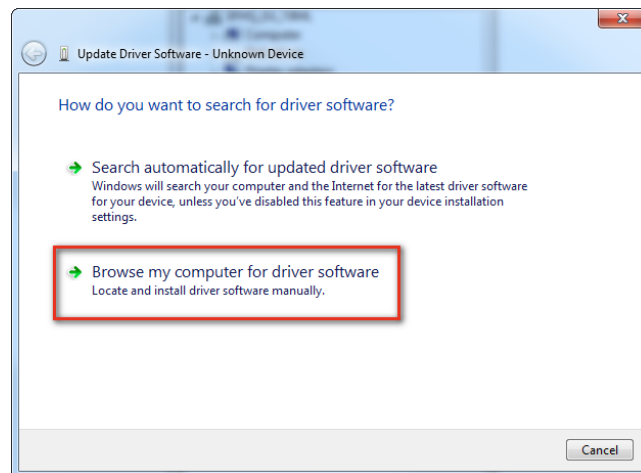

4. Click "Browse…" and navigate to the "../usbtinyisp_libusb-win32_1.2.1.0" folder you just downloaded. Then click **Next**.
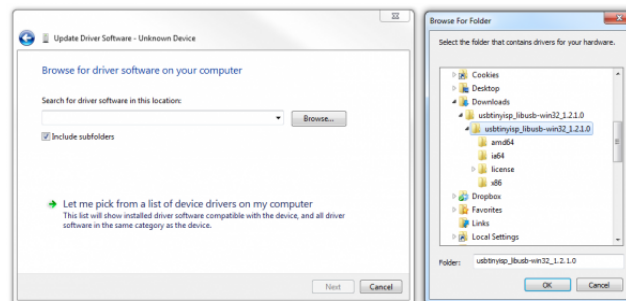
5. Windows will begin installing the driver, and then immediately notify you that the driver isn't signed. Click **Install this driver software anyway** option, to proceed with the installation.
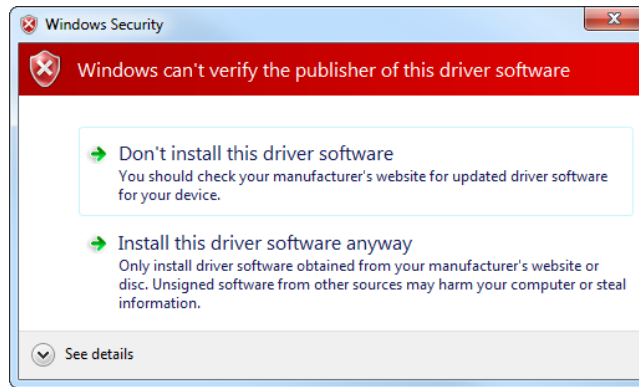
6. After a few moments, the driver should successfully install. You'll be prompted with a "Windows has susccessfully updated your driver software" window. Close that, and you'll see a "USBtiny" entry populated in the Device Manager, under the "LibUSB-Win32 Devices" tree.
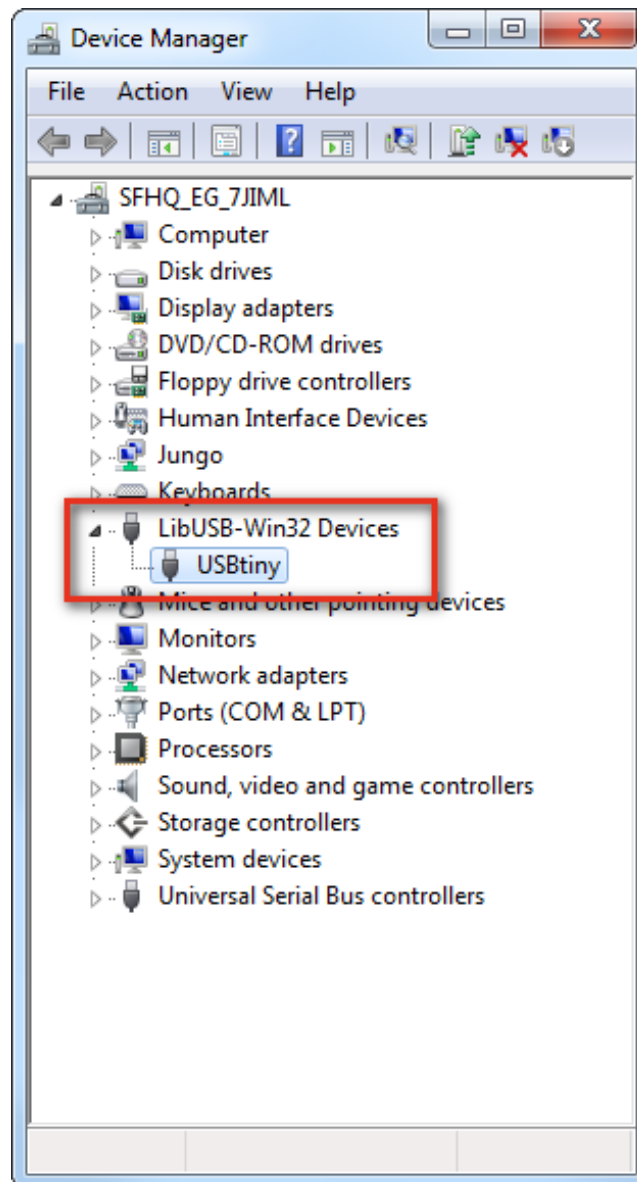
Congratulations! Proceed over to the next section, and we'll start using the Programmer!

Breathe easy now! Once you've installed the USBTinyProgrammer 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.
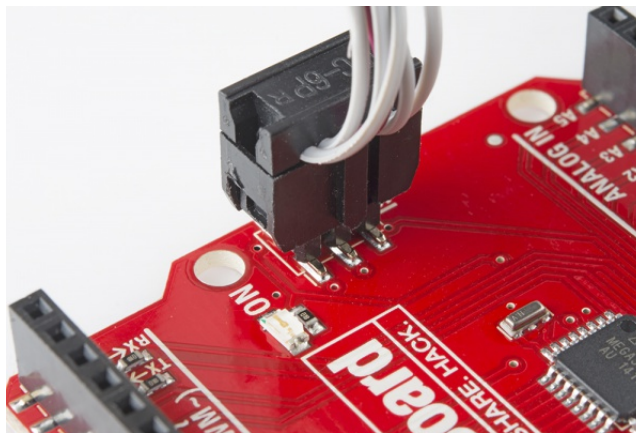

Note the notch on the connector facing the same direction as pin 1 (marked with a small white line here) on the 2x3 Arduino connector.

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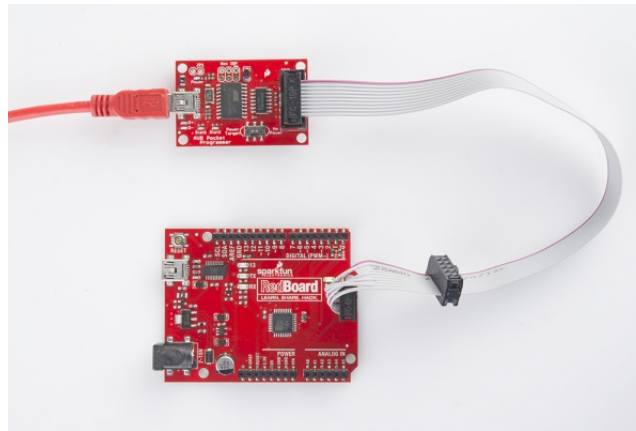
The "Power Target" feature is especially useful if you only have one USB slot/cable available.

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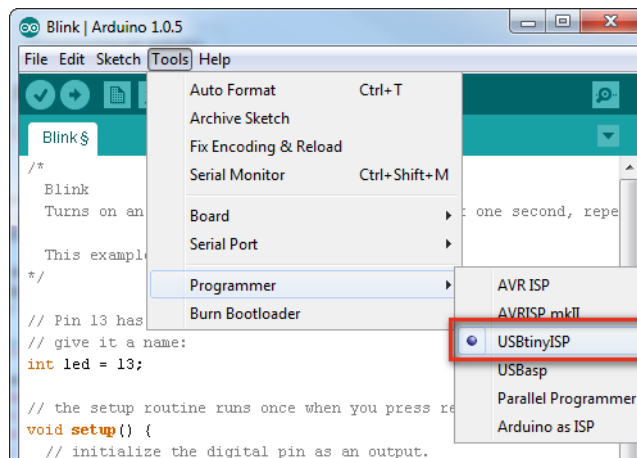
Also make sure you've **set the "Board" option** correctly! The serial port selection isn't required for uploading the sketch, but is still necessary if you're doing anything with the serial monitor.

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).
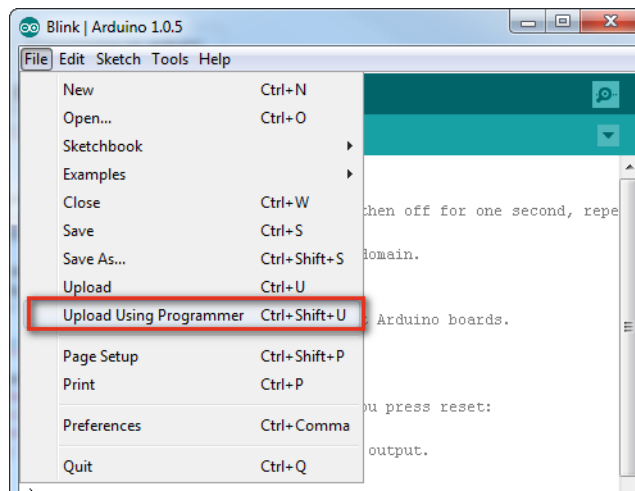
The Arduino will run through its normal process of compiling. After the sketch compiles, the Programmer will start lighting up blue everywhere – the "D+" and "D-" LEDs will light up, and so will the "Stat2" LED. When the "Stat2" LED turns off, the upload will be finished. Check the status area of your 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.
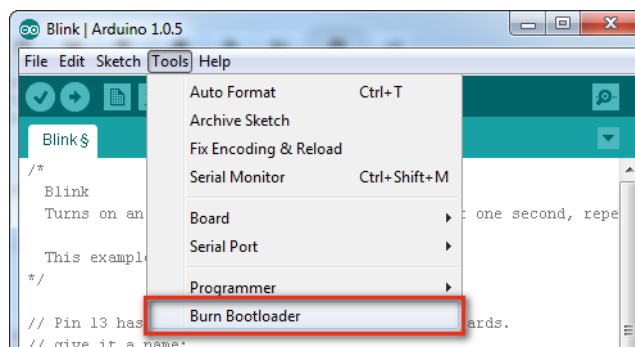
This process may take a minute-or-so. Not only will the bootloader be written into the flash of your AVR, the fuse bits (setting the clock speed, bootloader space, etc), and lock bits (barring the bootloader from overwriting itself) will also be (re)set.

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.
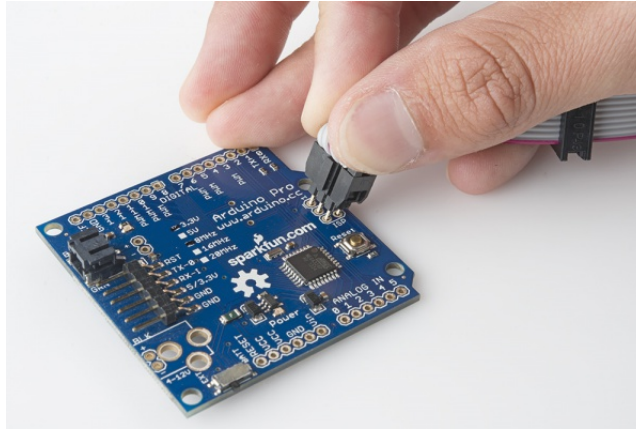
Another solder-less option is to use the ISP Pogo Adapter, which will afford you a more reliable electrical connection.
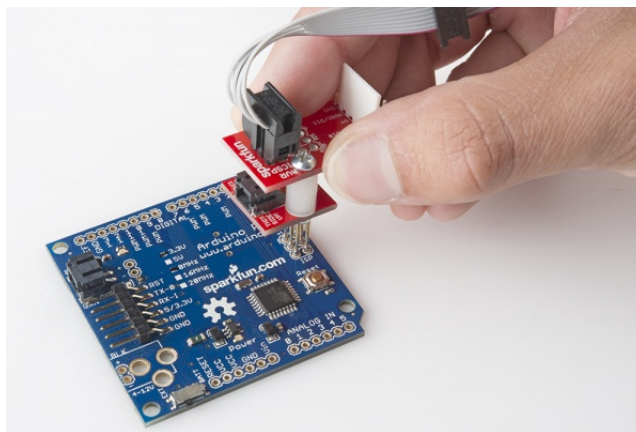
Both of these methods can be tricky – you have to hold those pins steady while the code uploads to your Arduino – but they're a good solderless, temporary option.

Using AVRDUDE

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 AVRs.

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 download documentation.
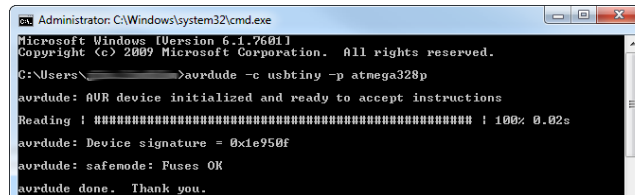
Sanity Check – Device Signature Verification

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, and your AVR Pocket Programmer is connected correctly, it's good to do a little sanity check first. Type this into your command prompt:

```
avrdude -c usbtiny -p atmega328p
```

(*Note*: This 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 AVR Device section below.)

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

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` .
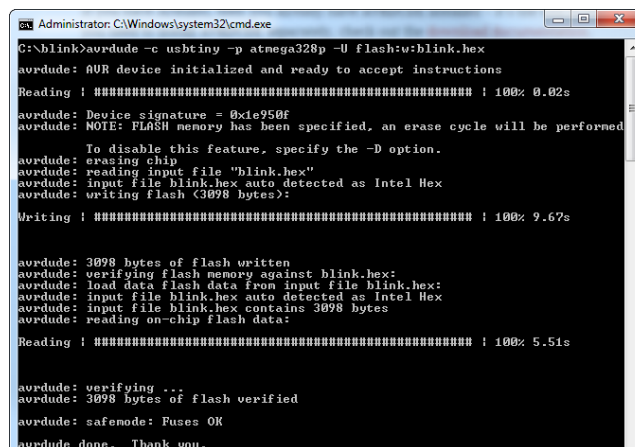
Flash Programming

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.

This command will perform a basic write to flash (using this HEX file as an example):

```
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.

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.

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".

```
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.

Useful Options

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

Specify 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 Option Documentation for the entire list of commands.

Resources & Going Further

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

- AVR Pocket Programmer GitHub Repository – Here you'll find everything from PCB design files, and firmware to custom enclosure designs.
- AVR Pocket Programmer Schematic
- AVR Pocket Programmer Firmware
- AVRDUDE Manual

Going Further

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!

- 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!
- 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 XBee/AVR Bootloading – Use your AVR Pocket Programmer to upload a custom bootloader, then wirelessly program your Arduino.