Generation 3 RFID reader (V. 10)
THE HELPFUL FRIENDLY MANUAL

By Eli Bridge and Jay Wilhelm

This manual provides general guidelines for getting started with the Arduino-based RFID reader for the University of Oklahoma Biologging Initiative.

Contents:

Welcome aboard!!

If you are reading this for the first time you are about to join a resourceful and creative group of people who apply RFID technology to interesting and impactful research. We are glad to have you. The Generation 3 RFID reader and it's predecessors have been used by dozens of research groups all over the world. Each of these applications have met with challenges and frustration, but there has always been a path forward. If you are new to RFID, Arduino, or electronics in general, you should expect to encounter some difficulty. We've tried to keep things as simple as we can, but we cannot anticipate every problem that can arise from every system or application. But you are not alone. There's a community here to help you (see the last chapter). There's also a helpful friendly manual, but you know that already.

Hardware layout and features

So you have a circuit board with a bunch of stuff on it. This chapter will give you a general ideas of what that stuff is and what it does.

First of all, what you've got is a type of Arduino. Arduino refers to a family of devices (circuit boards and accessories) and programming resources that are widely used by electronics enthusiasts. Because your RFID reader is an Arduino, you have a ton of resources available to help you get your research project (or whatever) going. There is a special Arduino programming platform or IDE (Integrated Development Environment) for Arduino that you can use to build your firmware and load it onto the circuit board. There are tutorials to teach you how to program using the Arduino IDE. There are hundreds of hardware accessories that you can attach to your RFID reader to allow you to do things like collect environmental data or make motors spin. And there are an equivalent number of software libraries that you can incorporate into your code to add functionality to your application. Just about anything that applies to Arduino in general can be applied to your RFID reader. However, there are different types of Arduinos, and they have different capabilities. **The RFID reader is modeled after the Arduino**

M0. So accessories and code that work with the M0 should work with your device.

Figure 1 shows the circuit board layout and identifies the key components. Here's what these components do:

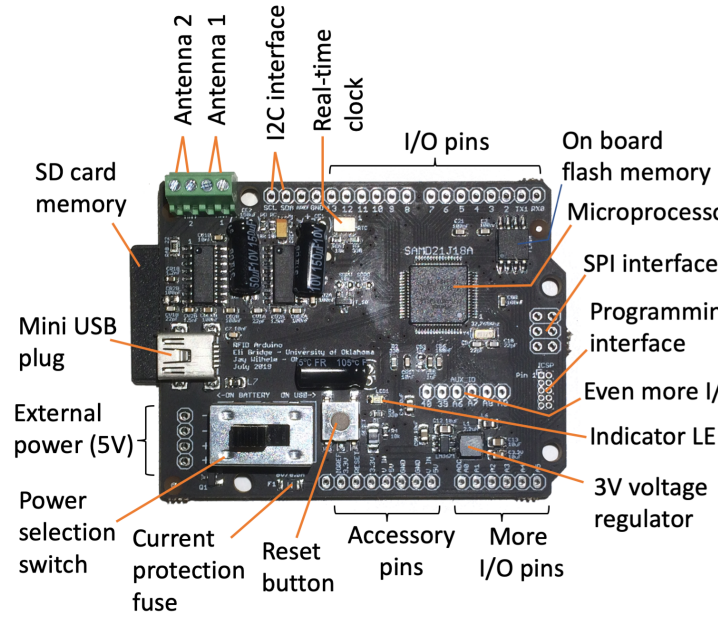


Figure 1: Major components

Microprocessor (SAMD)

This is the "brain" of the circuit board, and it is what you program when you upload. As microcontrollers go, this is a fairly sophisticated piece of equipment. If you want the whole story you can read the 300 page manual:

www.microchip.com/wwwproducts/en/ATsamd21g18

The short story is that this processor can do lots of things like:

- Interface with USB
- Carry out common communication protocols (serial, I2C, and SPI)
- Direct the activity of a few dozen input/output pins
- Perform analog to digital conversions (*i.e.*, measure voltages to take readings from external sensors and the like).
- Employ various low power sleep modes.
- Keep time (to some extent).

To get the processor to do these things you have to write some specialized code and load it up to the processor (but don't worry, you don't have to do this yet). Many of the input/output pins on the processor are connected to holes that line the edges of the circuit board (see "Headers" below). So you can hook things up with relative ease. Over the past few years there have been a few different hardware versions with different processors. The very first GEN3 RFID readers (v. 9.3) used a SAMD21. The current hardware models (v. 10.x) will feature a SAMD21J18A, SAMD21J17A, or SAMD21J17D. You can use the same firmware with these three different processors, but you have to make sure you use the correct board definition during programming (see the section below on loading the firmware).

RFID reader Integrated Circuit (EM4095).

This little thing does all of the work that enables RFID reads to happen. It generates and amplifies the carrier wave that energizes the tags, and filters the magnetic interference that

makes up the tag's response to the carrier wave. It outputs this filtered signal to the microprocessor, which then interprets the signal to make sense of the 1s and 0s coming in. There can be one or two of these on each circuit board. If you have two of them, you can alternate between signals from either one, but you can't have both of them active at once.

Real time clock (RV-1805-C3).

This tiny device is a real time clock with a built in crystal oscillator. It can operate with very little current (60 nanoamps) and it can be used to set alarms for waking the processor from sleep mode. It has a backup battery on the underside of the board so it can keep time even when the board is not hooked up to USB or a battery.

SD card – primary memory storage.

Data logged by the reader are stored on an FAT formatted SD card. You can easily grab your data by plugging the SD card into a computer and transferring the text file. Just about any FAT formatted standard size SD card should work.

Backup flash memory (AT45DB321E).

In case something goes wrong with the SD card, there is a built in backup memory. The capacity is 32 Mbits or 4,000,000 bytes. That's about 200,000 data reads, if you are just storing the RFID tag ID and a timestamp. A portion of this memory can also be used for other things like a reader ID number and operation settings.

Power inputs.

You can power the RFID reader either through the USB port or using the alternative power input between the USB socket and the power switch (see Figure 1). In either case you have to supply 5V. If you want to use a different voltage you will need to use an external voltage regulator to generate 5V.

The necessary ingredients.

For a full tag reading set up, you should have the following:

- Power supply (5V – this can be a USB port, battery, or phone charger)
- 1 or 2 Antennas (~1.25 milliHenries)
- Tags (EM41XX compatible or ISO11784/5 compatible)
- A USB cable with a "mini" plug on one end
- A computer running the Arduino IDE.
- An SD card formatted FAT32 (note that one card can service multiple readers)
- A coin cell battery for maintaining the clock (CR1025 or CR1632 depeding on hardware version).
- Small, flat screwdriver with a 1/16 inch blade.

If your reader is working and has firmware uploaded, you should be able to apply 5V, turn it on, scan tags, and see the LED on the circuitboard flash when a tag is read. To do more

than this basic testing you need to interface with a computer and customize a few things in the firmware.

But first there's some hardware set up to do. Fit the coin cell into the battery holder on the back of the circuit board. The positive side of the battery should face away from the circuit board. Attach To attach the antenna(s), use the screw clamps shown in Figure 1 and your tiny screwdriver. It does not matter which way round you attach the leads, but be sure to keep the inputs from antenna 1 and antenna 2 separate. For now you can either put the SD card in the socket or leave it out.

Getting things started.

Loading the Firmware.

OK, now for the hard part. To program and interface with the circuit board you will need to do the following:

- Download and configure the Arduino IDE to your computer
- Add the ETAG RFID board definition to the IDE
- Open some firmware in the Arduino IDE
- Connect the circuit board to your computer with a mini-usb cable
- Make sure the IDE "sees" your circuit board
- Upload firmware to the circuit board
- Establish serial communication with the circuit board
- Set the clock and any other relevant parameters
- Observe data logging activity through the serial monitor in the Arduino IDE

Step 1: Download and configure the Arduino IDE to your computer

First you must steer your web browser to the Ardunio website and download the Integrated Development Environment (IDE). It's here:

https://www.arduino.cc/en/Main/Software

Do not download any beta versions unless you enjoy suffering. Also they will try to sell you their online IDE, which you can try if you want. But I'm not familiar with it and neither is the Helpful Friendly Manual. Download and install following the website instructions. Once that's done go ahead an open the program.

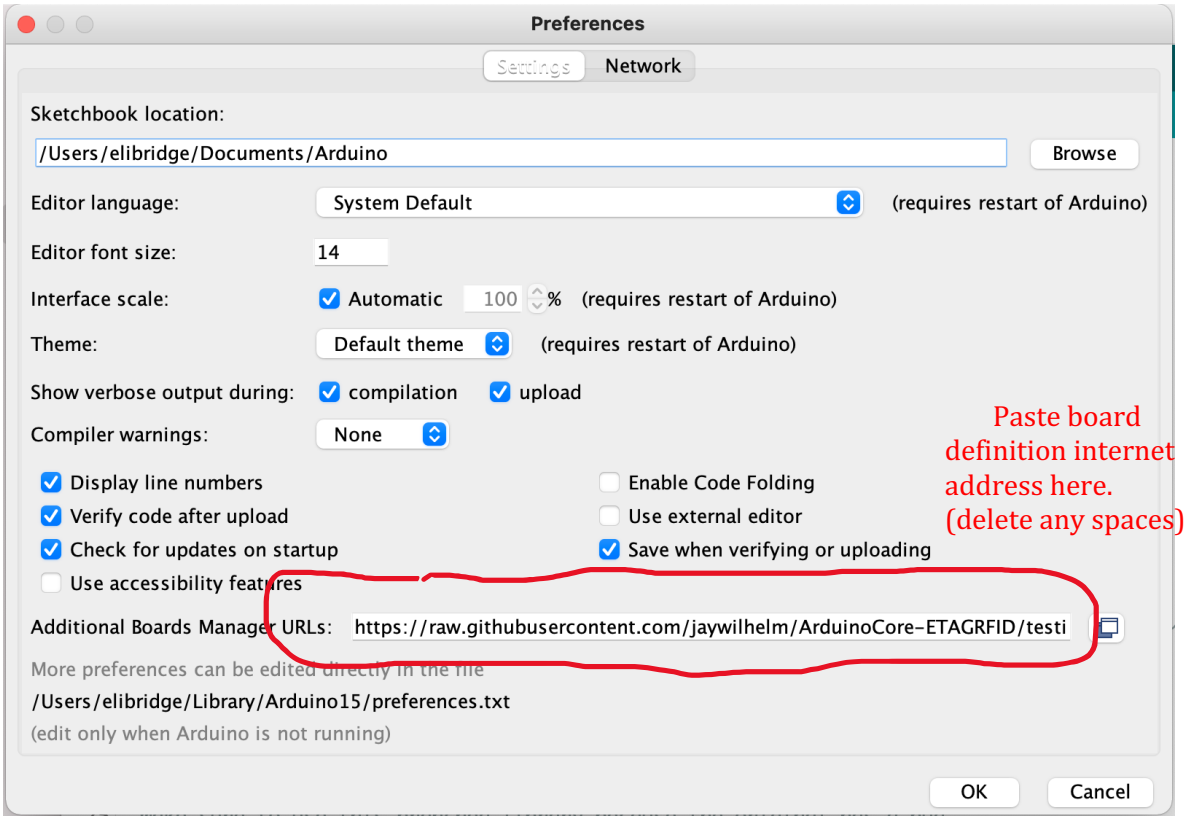
Step 2: Add the ETAG RFID board definition to the IDE

This is a little more tricky. For the Arduino program to work with a circuit board, the program needs something telling it what kind of processor is on the board and how the processor is connected to other components and pin headers on the board. This 'something' is a set of files referred to as a board definition. The Arduino IDE has all of its board definitions buried in a deep, dark, nearly-impossible-to-find directory, and that directory often changes when there new versions of the software. Fortunately, you can load a new board definition via the internet without manually moving files around.

1) The board definition you need is on the internet here:

https://raw.githubusercontent.com/jaywilhelm/ArduinoCore-ETAGRFID/testing/package_ETAGRFID_index.json

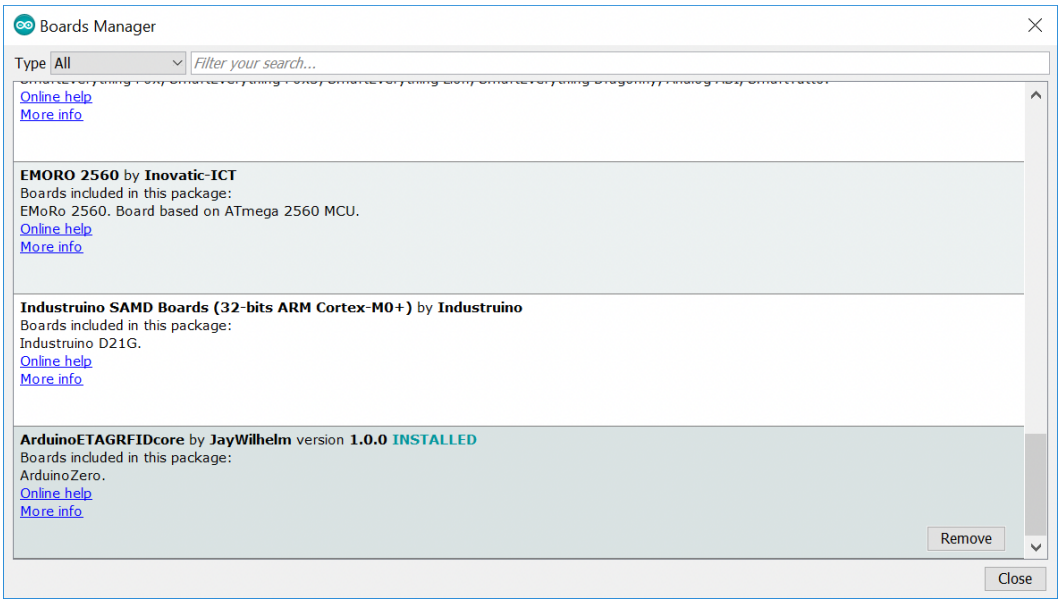
First you have to tell the Arduino IDE where to look for it: **File->Preferences (paste the URL into 'Additional Boards Manager URLs:' Make sure there is not a space after "ArduinoCore-")**



2) Tools -> Board: "xxxxxx" -> Boards Manager...

3) Scroll down to the bottom, click on ArduinoETAGRFIDcore, then click the 'Install' button in bottom right

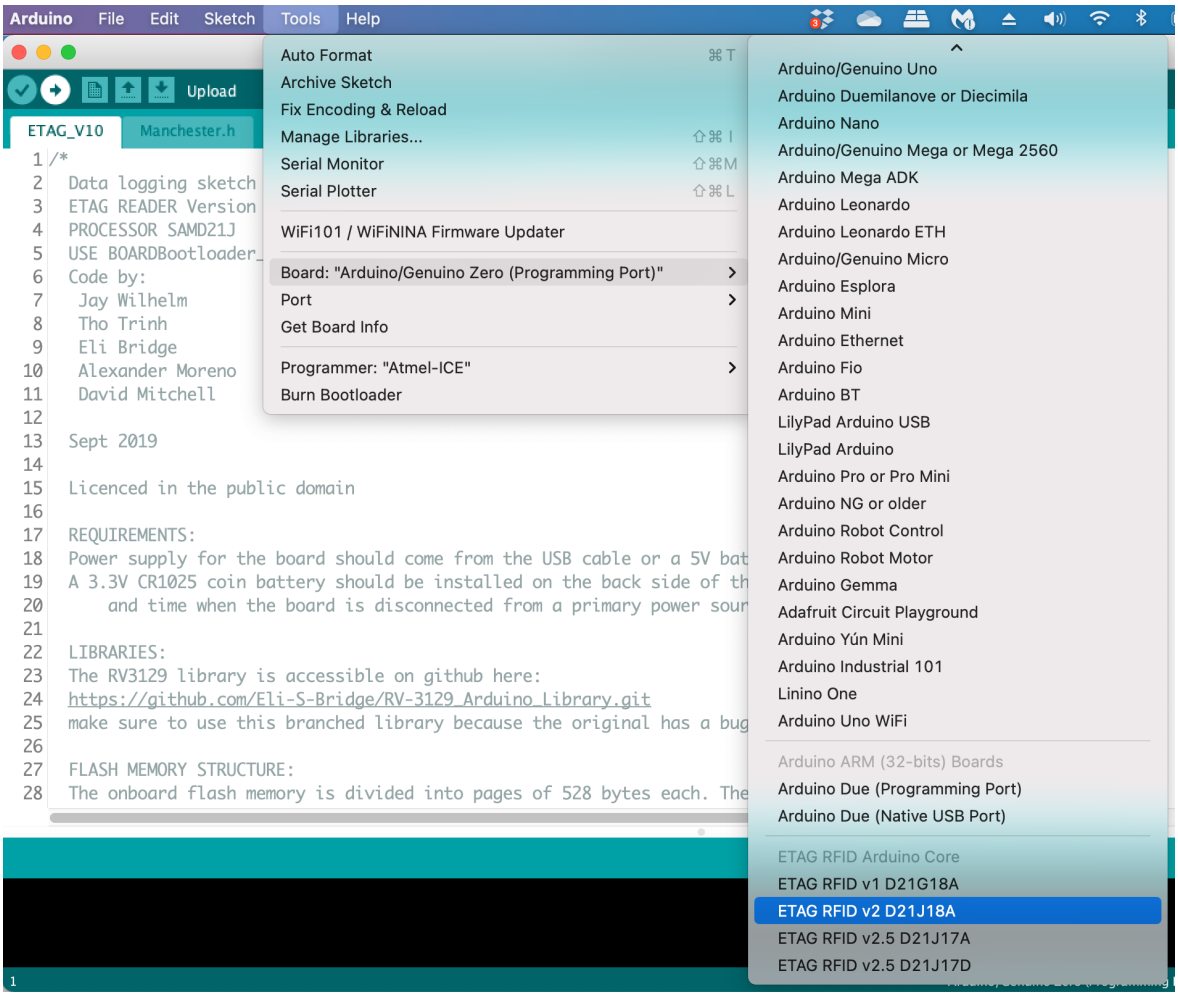
4) After installation check to see the word "INSTALLED":



5) Close the installation window and select the correct board definition. For the black circuit boards this will be “ETAG RFID v2 D21J18A.” If you have a gray circuit board it will probably be “ETAG RFID v2.5 D21J17D”. Green circuit boards could be “ETAG RFID v2.5 D21J17D” or “ETAG RFID v2 D21J18A.” The best way to make a determination is to read the part number printed on the microprocessor (see Fig 1). The last few characters on the part number should match the end of the board definition. Unfortunately, the print can be small and very faint. For the older blue circuit boards you should select “ETAG RFID v1 D21G18A.“

!!!!!Be sure that you don’t program a RFID reader with wrong” board definition!!!!

If you do the board will probably stop working. Using the wrong board definition damages the bootloader which enables UBS programming. You need a programming device (like an ATMEL ICE) to fix this, and you probably don’t have one.



6) Open a bottle of wine and celebrate your ability to follow instructions.

Step 3: Open some firmware in the IDE

Now you want to grab an Arduino program or "sketch" and load it onto your circuit board. The default sketch for version 10 is here:

https://github.com/Eli-S-Bridge/ETAG_V10

If you are familiar with Github, then you can clone the directory and all that jazz. If you are not a Github elitist then look for the green button that will let you download a zip file. Unzip that file and rename the folder so it has the same name as the file that ends with .ino (minus the .ino). Now you can open that file in the Arduino IDE.

Step 4: Install the clock library

You also need to install a library for the real time clock. This library is also on Github:

https://github.com/Eli-S-Bridge/RV-3129_Arduino_Library

Again, download the zip file by clicking the green button. Once the folder is downloaded and unzipped, rename the folder to "RV-3129" and put the whole folder into the libraries folder within your Arduino folder. The Arduino folder is created when you install the Arduino IDE and it is the default location for saving sketches. It's usually found in your Documents folder.

Step 4: Connect your circuit board.

Plug in the circuit board with the USB cable and push the slide switch on the board to the USB power position (toward the inside of the board). Next we have to make sure the Arduino program is playing nice with the circuit board. Click on tools → port, and see if there is something there that looks promising. Things like "Bluetooth" and "Iphone" are not promising. Something that includes "M0" is promising.

Step 5: Upload the firmware

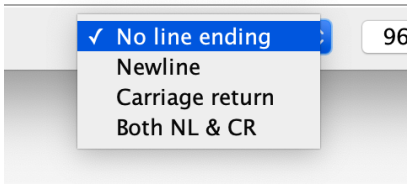
!!!!!REMOVE THE SD CARD!!!!!!

It is strongly recommended that you do not have an SD card installed when you upload firmware to the board. If you happen to have some SD card write instruction going on, the firmware upload could interrupt things and mess up the SD card file structure. So pull out the SD card at this point if you need to. If everything is working to this point then you should be able to click on the right pointing arrow at the top left of the Arduino sketch window to compile and upload the firmware. This will take several seconds. Depending on your Arduino IDE setting you should see some degree of feedback in the messaging window below the code. If the IDE says "Thank you." then it worked.

Establish serial communication with the circuit board

As soon as the firmware is uploaded the circuit board will start running the program. You have a few seconds here to establish a serial communication connection so you can interface with the board. To do some communicating activate the serial monitor in the Arduino IDE (Tools→Serial Monitor). There's a text menu transmitted from the board when it first starts up. But if you are too slow to start the serial monitor, you may not see it.

At the bottom of the Serial Monitor window is a pull-down list for appending characters to the text you send the RFID reader. **Be sure to select "no line ending" as shown here:**



If all is going well you should see a text menu that looks like this:

```
    00/02/2010 18:17:48
    Device ID: A101
    Logging mode: F (Data saved to flash mem only; no SD card
logging)
```

```
What to do? (input capital letters)
    C = Set clock
    B = Display backup memory and log history
    E = Erase (reset) flash memory
    I = Set device ID
    M = Change logging mode
       W = Write all flash data to SD card
```

This menu lists the commands you can use to change the settings on the RFID reader. To send a command, click on the input line at the top of the serial communicator window. Then type the letter (capitals only please) and hit enter. If you send something other than the letters listed on the menu the reader will just start logging data. To get back to the menu send a lowercase "m".

Let's go through the menu commands shall we?

C -> Set the clock.

If you enter a "C" you should see a prompt that says "enter mmddyyhhmmss." When setting the clock follow these rules:

- Enter dates American style with the month first
- Use leading zeros for 1 digit values
- Use military time

So February 8, 2023 9:27:06 would be entered as "020823092706." After you hit return you should see the menu again with the updated date and time.

B -> Show memory contents

If you enter "B" the reader will transmit all of the data in the flash memory in human readable form. So you can see what data the reader has stored, and if you wish you can copy the data and paste it elsewhere. The RFID tag logging data is displayed first followed by the log messages.

E -> Erase

When it is time to start over with a reader, enter E to erase all of the flash memory. This memory is full erased (*i.e.*, overwritten) So make sure any existing data are saved elsewhere. After entering "E" you will be prompted to enter the word "ERASE" as a safey precaution. Do this and your data are gone forever.

I -> Device ID

It's a good idea to give each reader its own special name. This name is used when data are transferred to an SD card (The reader creates a text file on the SD card with a filename that includes the device ID). When you enter and I you will be prompted to "Enter four alphanumeric characters." Choose some combination of four capital letters and numbers and give your reader a name!

M -> Change logging mode

Once data logging begins, data can be stored in two places—an SD card (if one is available) and a backup flash memory chip built into the circuit board. The default mode is flash-only mode (Mode F). In this mode you can log data to just the flash memory. The alternative mode (Mode S) writes data to both the SD card and the backup flash memory as

soon as a tag is read. Of course, you need to have an SD card inserted into the socket for this to work. If you are running in S-mode and the reader fails to detect an SD card you will see a warning that precedes the text menu and a long series of LED flashes on startup to indicate the SD card is missing. By entering "M" at the menu you can toggle between logging modes.

In either logging mode, if the reader detects an SD card on startup it will transfer any new data on the flash memory (i.e. data not already stored on the SD card) to the SD card. So to get data on to an SD card, just insert the card and turn the reader on. The reader will transfer tag-reading data to a file called "XXXXDATA.TXT", where "XXXX" is the device ID. All log messages will be written to XXXXLOG.TXT. If one of these files does not exist on the top directory of the SD card, then it will be created. If the target files already exist, then any new data flash memory will be appended to the end of the existing text file.

W -> Dump the flash memory to the SD card.

Entering a "W" at the menu will transfer all of the data from the flash memory to an SD card. The reader will use the same file names described above. If the files exist then ALL data in the flash memory will be appended to the ends of the two files. This process may result in duplicated lines in the SD card file. If the target files do not exist, they will be created and populated with text.

Observe data logging activity

Start the reader logging by sending it a random letter (or just wait a bit and it will start on its own). Now you should see a bunch of time stamps scrolling down the serial monitor window as the reader attempts to read tags. If you have an antenna plugged in and a tag handy you can try to get the tag to read—Oh looky…RFID data coming in. Have fun with this for awhile. If you want to go back to the menu just enter a lowercase "m". If the serial stream suddenly stops, don't worry. Just reference the part about "cycleCount" in the troubleshooting section below.

Customizing the firmware

Basic data-logging parameters

OK. Enough fun. Time to set up the firmware so the reader does exactly what you want it to do. This means delving into the C and C++ code that makes up the Arduino programming system. But don't worry. You can do this. One thing that will help is code annotation. There are lots of notes in the code all written in plain English. They are there to help you interpret what the code is doing. Individual notes are defined by a double slash ("//") at the beginning of the statement. Larger blocks of notation start and end with "/*" and "*/" respectively.

If you just want to set up a simple data logger you just need to tweek a few settings. The most important things are setting sleep and wake times and establishing a polling cycle to detect tags. The core Arduino code for the ETAG reader will evolve over time, so I cannot tell you what line numbers to go to. But somewhere near the beginning you should be able to find a note that says something like:

/***CONSTANTS (SET UP LOGGING PARAMETERS HERE!!)***

Below this line are a bunch of variables (in this case constant or unchanging variables) that get defined. For example

```
const byte checkTime = 30;    // How long in milliseconds to check
      to see if a tag is present (Tag is only partially read during
      this time -- This is just a quick way of detirmining if a tag
      is present or not

const unsigned int pollTime1 = 200;   // How long in milliseconds
      to try to read a tag if a tag was initially detected (applies
      to both RF circuits, but that can be changed)

const unsigned int delayTime = 8;   // Minimim time in seconds
      between recording the same tag twice in a row (only applies
      to data logging--other operations are unaffected)

const unsigned long pauseTime = 100;   // CRITICAL - This
      determines how long in milliseconds to wait between reading
      attempts. Make this wait time as long as you can and still
      maintain functionality (more pauseTime = more power saved)

uint16_t pauseCountDown = pauseTime / 31.25;  // Calculate
      pauseTime for 32 hertz timer

byte pauseRemainder = ((100*pauseTime)%3125)/100;   // Calculate a
      delay if the pause period must be accurate
//byte pauseRemainder = 0 ;   // ...or set it to zero if accuracy
      does not matter

const byte slpH = 99;  // When to go to sleep at night - hour
const byte slpM = 00;  // When to go to sleep at night - minute
const byte wakH = 99;  // When to wake up in the morning - hour
const byte wakM = 00;  // When to wake up in the morning - minute
```

Notice all the helpful annotation comments that follow "//" on each line of code? Terms like "`const byte`" are telling the program that you want to establish a constant that is stored in a byte-size memory location. "`const unsigned int`" means the same thing except now it is an unsigned number (only positive values) and it is stores in an integer memory location, which allows for two bytes. The first thing to concern yourself with is the constant pauseTime. As the annotation implies, pauseTime is a time value in milliseconds and it is the main determinant of the tag polling cycle. If you make pauseTime a large number like 5000, then a full five seconds will elapse between attempts to read a tag. If you want to monitor something that can go in and out of read range in a second or two, you will miss a lot of activity if pauseTime is set to 5000. However, having a long pauseTime helps save power. So you want to have the longest pauseTime you can get away with. Once you decide what pauseTime you want to try, type it in after the equals sign. When you reprogram the RFID reader this new pauseTime constant will go into effect.

`CheckTime` determines how long to do an initial check for a tag. I don't recommend changing this much unless you have plumbed the depths of the tag reading functions and know what you are doing. In a nutshell, during each tag polling cycle the reader spends a short amount of time (`checkTime`) just checking to see if it is getting pulses that are consistent with the presence of a tag in the reading field. If the pulses are not appropriate (too long or two short in duration), then the reader proceeds to sleep for the duration of `pauseTime`. If a tag is detected, the reader continues to monitor the pulses and attempts to decode them into a viable tag ID. The constant `pollTime1` determines how long the reader should keep trying to read tags if it detects something promising during `checkTime`. Again this probably does not need adjusting for most applications.

Finally, `delayTime` determines how frequently you want to log data for the same tag over and over again. If a critter is just sitting on top of the antenna so that it's tag is constantly in the read range, how often do you want to log a line of data. The value is in seconds so the most frequent recording you can do is once per second. Note that delay time only applies to instances where the same tag is read over and over. If a different tag is read or if a tag is read on a different antenna, `delayTime` does not apply—you will get a new data line recorded for the tag.

Another important set of constants determine long term sleep parameters which can be employed if you want to put the reader to sleep (and save power) during certain times of day. These constants (`slpH`, `slpM`, `wakH`, and `wakM`) determine the hour and minute for when the reader should go to sleep and the hour and minute for when the reader should wake back up.. As the reader goes to sleep at the designated time of day, the real time clock sets an alarm based on `wakH`, and `wakM`. and that turns everything back on when it's supposed to.

Advanced customization

OK, you ask, what if I need the RFID reader to make Belgian Waffles using a propane torch and a tire iron. Well, that might actually be possible, but you will have some work to do. The RFID reader can certainly do anything you want it to with regard to fancy logging-schedules, reading data from other sensors, recording additional data or messages, and controlling devices like motors and solenoids. But we cannot accommodate everyone's wishes in a single program. If you want to implement these sorts of things you will need to learn a bit about how to program in the Arduino language and then figure out how to incorporate your additional functions into the logging scheme. There are lots of resources to help you and there should be several example programs you can draw from. But there's no escaping the need to delve into the Arduino language a bit—unless you can employ a technician to do it for you. A Google search will find you lots of resources for learning Arduino. Here are a few of them:

- The Arduino website (www.arduino.cc) – no brainer there.
- The Arduino language reference: https://www.arduino.cc/reference/en/ - first place to go when you see something new.
- The Arduino M0 website: https://store.arduino.cc/usa/arduino-m0 - The RFID reader is base on the Arduino M0. So most of what applies to the M0 applies to the RFID reader. Except of course the M0 does not have a built in clock, SD card, and RFID reader.

Antenna design and construction

Arts and crafts time. The RFID reader requires some sort of antenna. Usually this is a tight coil of magnet wire somewhere between 5 and 15 cm in diameter. Antenna coils are easy to make and can assume a wide variety of shapes and sizes. The trick is that you have to figure out how many turns are needed for your particular shape and size to get the correct inductance for your antenna. Inductance is a property of wire conductor coils that determine how well they resonate with a particular frequency of alternating current. Think of it like repeatedly striking a bell and trying to get as much sound as possible. If you strike the bell in time with the bell's resonant frequency, then you can maximize the sound output. It's sort of like that with the RFID reader except we want to maximize radiated energy output. The frequency of the RFID reader is set at 125 kHz, so we need to get the antenna to match up. The ideal antenna inductance turns out to be around 1.3 milliHenries (mH) in theory. In practice, 1.2 to 1.3 mH works pretty well. As you add more turns to a coil, its inductance goes up—but it is not linear and the amount of added mH per turn depends on the size and shape of the antenna. So for now, we have to make a guess about the number of turns, measure the antenna inductance, and then adjust the number of turns accordingly until we get it just right.

First of all, you are going to need some sort of inductance meter. Some general purpose multimeters have inductance functions built into them. Then there are LCR meters which are a bit more specialized for inductance and capacitance measurements. You just have to buy or borrow one of these. Sorry. A cheap one will do.

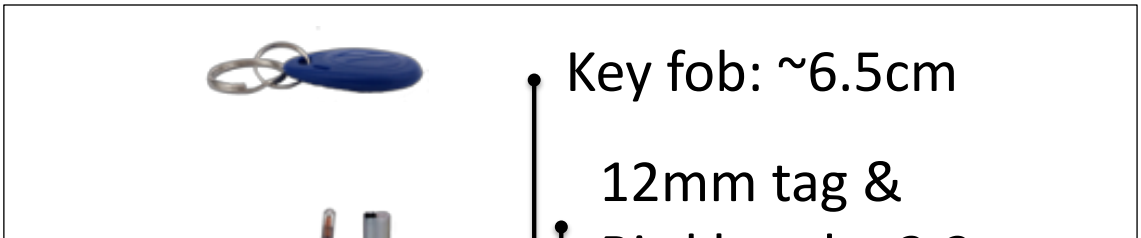
For best read range, I recommend a round antenna of about 10-12cm in diameter. To make a simple antenna, you simple wrap magnet wire (26 to 30 gauge) around a form or an object, while applying glue (waterproof fabric glue or nail polish) to stiffen the coil. When the coil is finished, remove the form and strip the leads from the coil. These leads should be soldered or clamped to the two terminals labeled "ANTENNA" on the circuit board.

For reference, a 7cm x 5cm rectangle requires about 103 turns, and a 12cm hexagon requires about 84 turns.

Tags

What tags should you use? There are lots of them out there. First of all the tags have to be EM41xx compatible. EM41xx refers to a communication protocol that defines how the tags send out their tag ID data and the parity checking information that helps prevent erroneous reads from being accepted. EM4100 and EM4112 tags will both work. Tags that say they are Trovan compatible or ISO11784/5 compatible will not work (although it is possible to modify the reader to work with these tags).

Among the EM41xx tags you will find a large number of shapes and sizes available. Generally speaking, you want to use the largest tag that is appropriate for your application. All 125kHz tags have antenna coils in them. Larger tags have larger antennas which means they can absorb more of the RF energy coming from reader. So, to increase read range you want the largest antenna you can get away with. Here are some examples of tag sizes and read range.



Antennas

RFID antennas are generally just a coil of magnet wire. Magnet wire is just a strand of copper with a non-conductive enamel coating on it. You can make a coil by wrapping wire around something non-metallic, like a chunk of wood or use a removable form to make an open coil for tags to pass through. You can also make a stick antenna by wrapping wire around a ferrite rod.

RFID antennas have to be tuned to the frequency they are meant to generate (in this case, 125 kHz). A simple way to accomplish this is to simply construct antennas that have a certain inductance. Inductance is a measure of how a coil responds to changes in current. As you add more coils to an antenna its inductance increases, and vice versa. For this particular RFID reader, we need an antenna with an inductance of about 1.3mH.

You can buy antennas if you want. A Canadian company called Qkits sells a small RFID antenna with an inductance of 1.26mH (close enough!!). There may be other vendors too. Make sure you know what the inductance is before you buy anything.

You can also design and make antennas. Making the first one is the hard part. You generally want to estimate the number of turns you will need (based on similar coils), make your loop with a few more turns than you estimated, measure the inductance, and then remove turns until you are close to 1.3mH. In other words, it's a trial an error process when you make the first antenna. After you've established how many turns you need

Some tricky issues you should be aware of:

1) The size and shape of the antenna affect the inductance, as do a number of other factors. Hence, it is difficult to provide a formula for calculating the number of turns needed to achieve a certain inductance. And so we have to rely on guesswork and trial and error.
2) The tightness of the coil will affect inductance. So if you plan to wrap the coil in tape or something, you may get different inductance measurements before and after the application of the tape. Generally speaking, you want to have an orderly and tightly coiled antenna for best performance.

Another issue with antennas is how to orient them for a particular application. Read range is greatest when the antenna coil has the same spatial orientation as the coil in the tag.

Troubleshooting

Some people live charmed lives and everything just works for them right out of the box. The Helpful Friendly Manual does not care about those people. They have it too easy. For most of us there are going to be problems. Scan the bold print below to see if your problem is addressed, and if not maybe try posting it to the RFID forum here:

https://ornithologyexchange.org/forums/forum/208-rfid-working-group/.

General tips.

- **Swap it out:** Sometimes problems are associated with a malfunctioning circuit board or accessory. So, one of the first things to do is to try swapping out different devices. If the problem is consistent across devices then you've got a real systemic problem and not an isolated issue.
- **Check hardware:** I know, I know, you already went over all of your connections and everything is securely connected and wired up. Just do it again anyway.
- **Verify the USB cable:** It's amazing how often these cables go bad. Check your cable with a device you know is working or swap it for a new one.

Common problems.

The Arduino IDE cannot connect to the bloody circuit board. The likely culprit here is that installation of the board definition failed. The things to check here are your internet connection and the url for the board definition. If your web browser is working, then you probably have a good internet connection. In that case make sure the url matches exactly in the additional board manager URL window. Sometimes when you copy and paste your computer throws a space into the text just to show you that it loves you. If correcting the url fails, try installing some other board definitions. If that does not work, you are probably looking at a reinstallation of the Arduino IDE.

The stupid sketch won't compile and/or upload. You really have to get the planets to align to make the firmware compile and upload. You need to have all the necessary libraries in the right places, you need to have the correct board definition, you need to be connected to board, etc. If you get some errors when compiling they can sometimes give you a clue as to what is wrong. To see all of the error output, go into the Arduino preferences and enable verbose output. Look through the error messages to see if you can figure out what is wrong. Some common issues are:

- **Missing Libraries** - If the errors are related to files not being available, then make sure you have all of your libraries in place (i.e. in the "libraries" folder). Libraries are chunks of code that have lots of functions that are meant to be used by Arduino sketches. Each line of code that starts with "#include" is specifying the use of a library. Some libraries are built in to the Arduino IDE. For example, Wire.h, SD.h, and SPI.h are built in libraries and they shouldn't be giving you any trouble. But there may be other libraries that have to be collocated with the main sketch (i.e. in the same folder), or loaded into the "libraries" folder in your Arduino code directory (the default location

for saving sketches). I can't offer you specifics because the libraries used are likely to change with different versions of the software. But putting things in the libraries folder is a good thing to try. You can also get missing libraries because file-transfer issues if you bringing the libraries in from another computer (e.g. via email or a thumb-drive).

- **Wrong board definition** - Your problem may also be that you have the wrong board definition selected. Unfortunately, the error messages never say useful things like "wrong board definition." But make sure you've got that bit correct. Seems silly to put this here but I still make this mistake all the time. Just now I spend 10 minutes fighting with this opaque error message: "SerialUSB' was not declared in this scope." I then changed the board definition and all was well. Using the wrong board definition can also mess up the bootloader, which makes the board impossible to program until you restore the bootloader with a special programming device (see page seven). But **before you give up on a board, try this: start uploading the sketch to the board and then quickly press the reset button** (small button near the on/off switch. That sometimes allows the sketch to load.

- **Missing files** - You may get an error for a missing file like "**arm-none-eabi-g++.**" For fancier Arduinos like the M0 and the RFID reader, there is an extra set of tools needed for the Arduino IDE to communicate with them. This package of tools should have come in along with the ETAG board definition. But if it didn't you can try going to the boards manager and install one of the "Arduino SAMD Boards" packages, (one which includes the Arduino M0). Then try uninstalling the ETAG board def and reinstalling it.

- **Text transfer** – If you attempt to copy code or library files from one computer to another, you might have problems, especially with Mac-to-PC transfers. Macs and PCs encode text differently, so you can get errors associated with spaces and tabs and such. This can even apply to zipped files. It's best to get everything directly from the sources (like github).

The damn clock keeps losing the correct time. Make sure the clock battery is secure. Battery connections are always potential weak points in electronics. If the small clock battery is not making contact at its positive and negative terminals, then the clock will lose the date and time when power is switched off. You can try cleaning the battery and the battery holder. You can also try bending the battery holder a bit to put more contact pressure on the battery. Using a multimeter to determine that the 3V from the battery is transferred to the metal on the battery holder is a good way to check for this problem. To do this voltage check, set your multimeter to read in the range of 3V, touch the black probe to a connection point labeled GND, and touch the red probe to the side of the battery holder. If you do not see about 3V on the readout, then you have a bad battery or a bad connection.

The reader stops sending serial data for no apparent reason. Ah, but there is a reason. To save power the reader only outputs serial data for a limited number of polling cycles. The reader cannot go into "deep sleep" and maintain serial communication. So after a while the reader stops sending serial data but it should continue to read tags and store data. There is a variable called stopCycleCount that defines the number of cycles before serial communication stops. You can change this number if you want serial communication to end sooner or later. Or to keep serial communication on all the time, find this line:

```
if(cycleCount < stopCycleCount){
```

and change it to:

```
if(1){
```

I can't set the clock! When I enter 'C' at the menu prompt, I immediately get a resonse that says: 'Time entry error.' This is probably because the serial monitor is sending extra characters (like a carriage return) when you transmit some text. Make sure you have "No line ending" selected at the bottom of the serial monitor window (see section on establishing serial communication above).