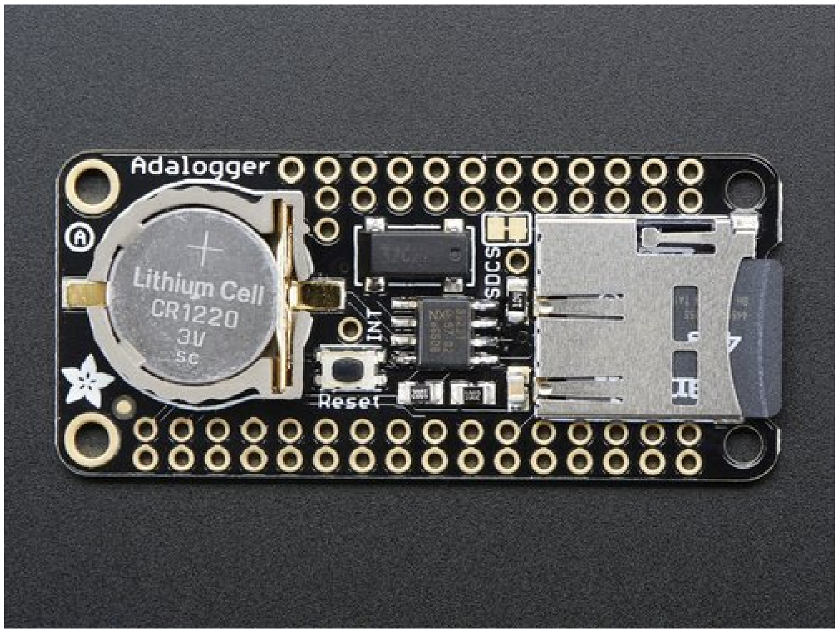
Adafruit Adalogger FeatherWing

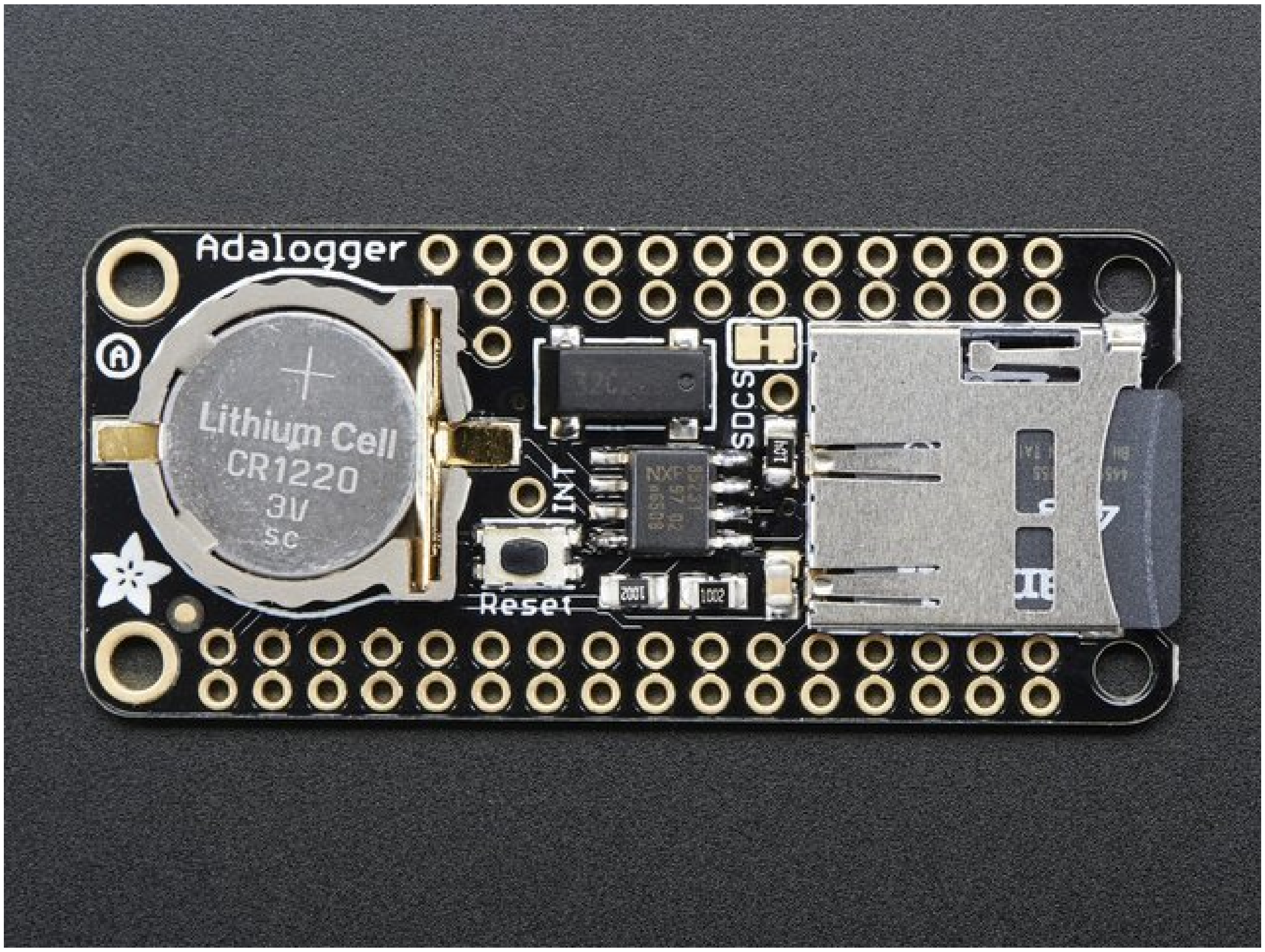
Created by lady ada



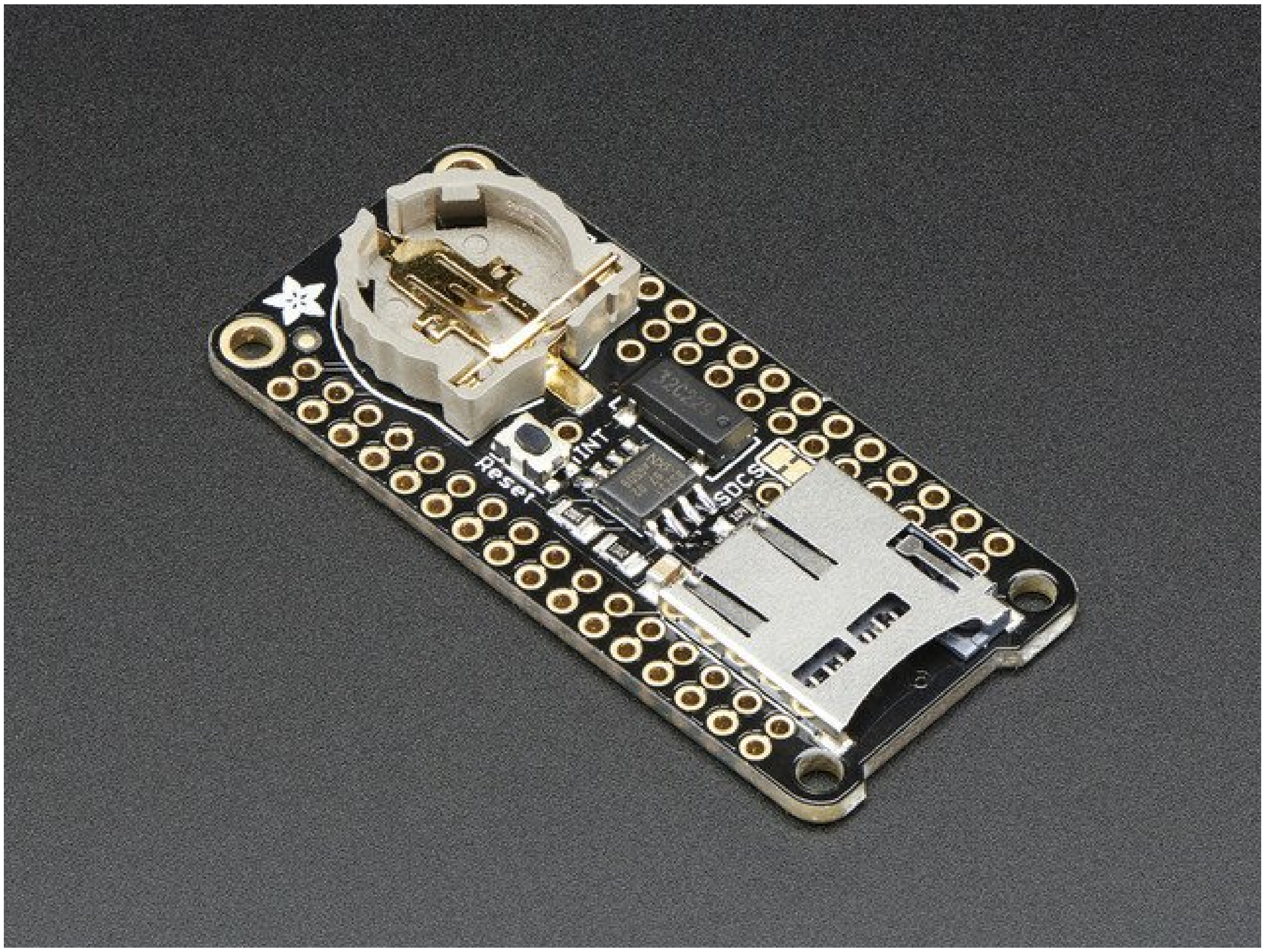
Last updated on 2016-09-07 06:18:59 PM UTC

Guide Contents

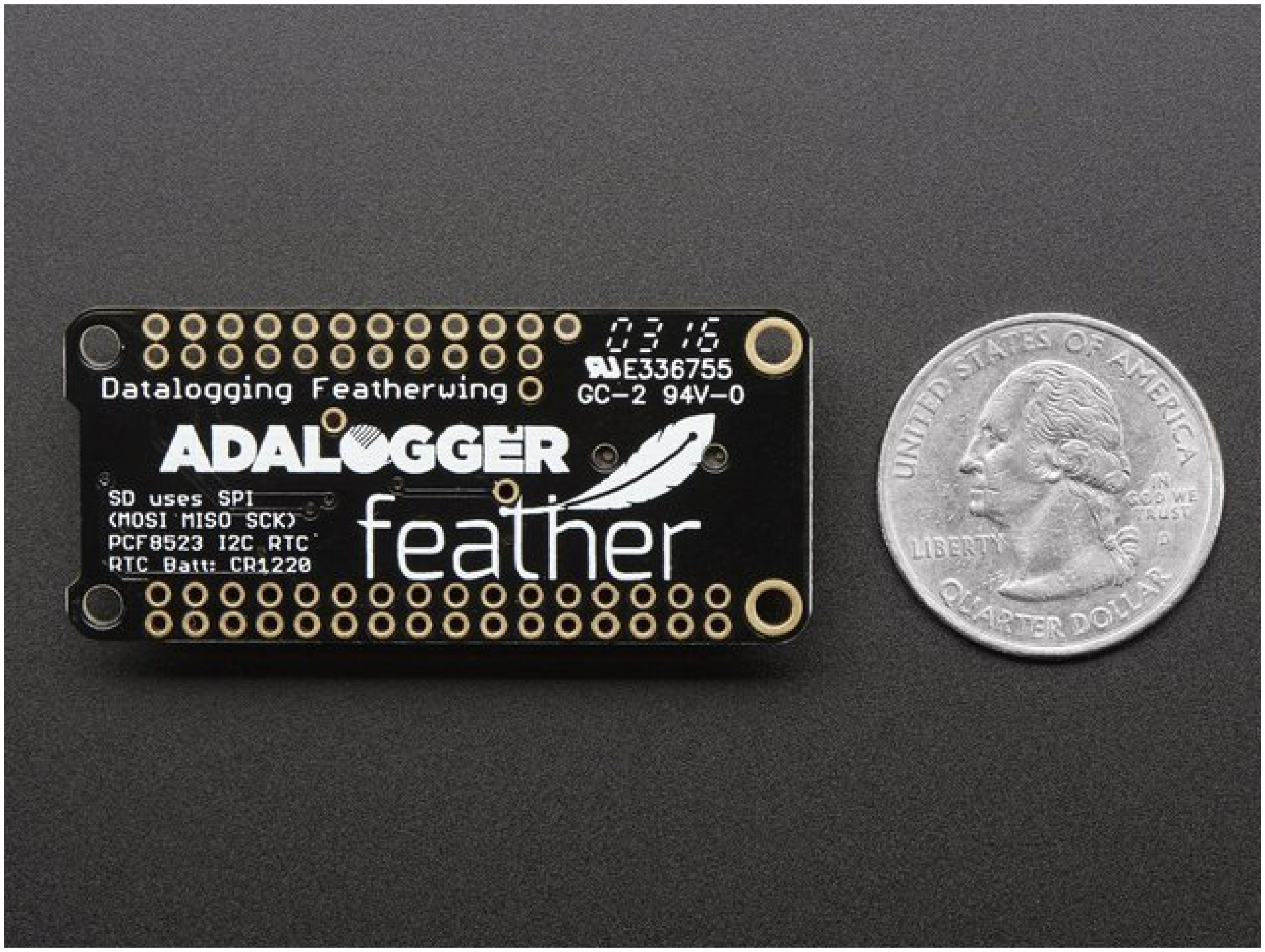
Overview



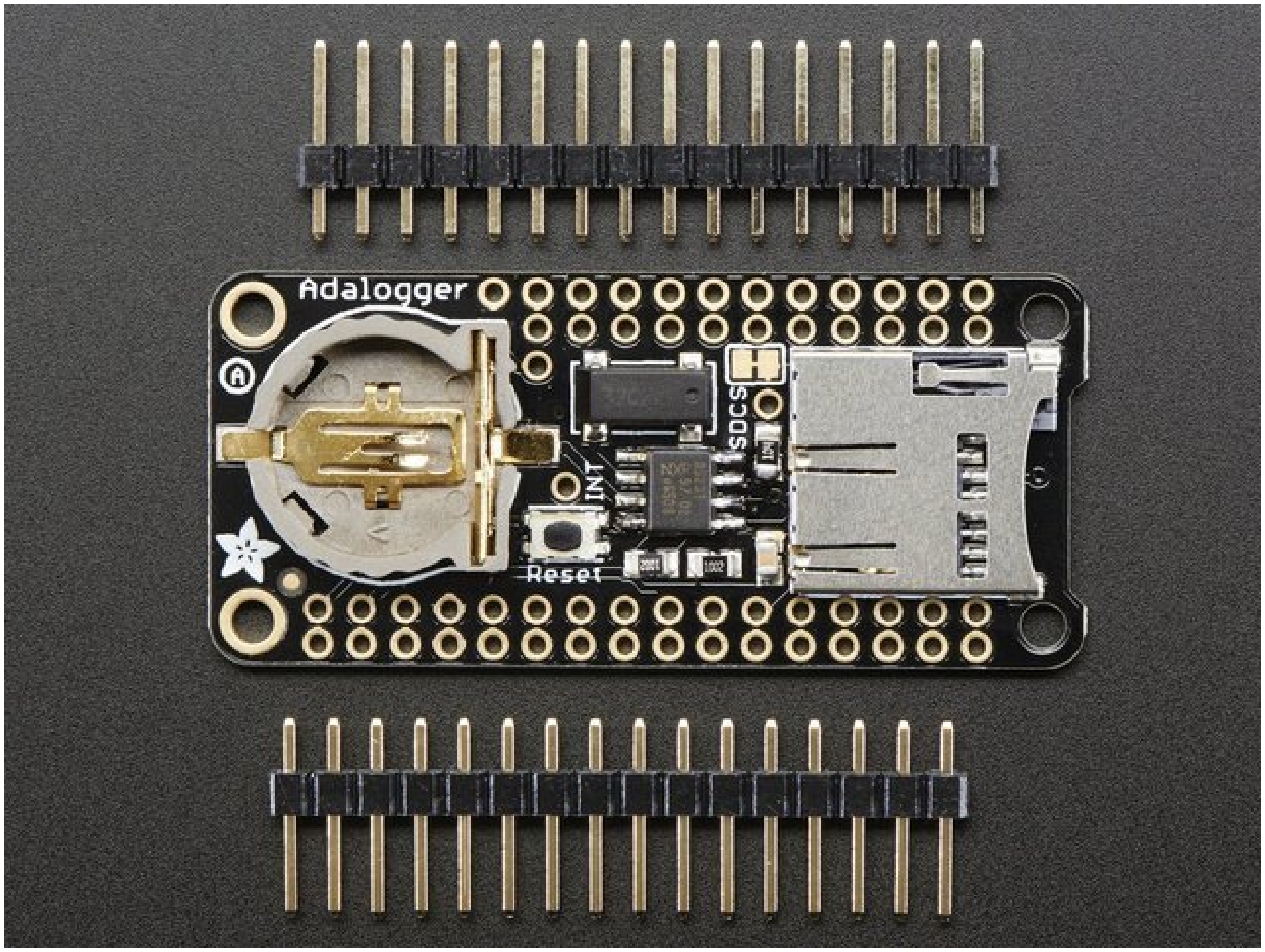
A Feather board without ambition is a Feather board without FeatherWings! This is the **Adalogger FeatherWing**: it adds both a battery-backed Real Time Clock and micro SD card storage to any Feather main board. Using our Feather Stacking Headers (http://adafru.it/2830) or Feather Female Headers (http://adafru.it/2886) you can connect a FeatherWing on top of your Feather board and let the board take flight!



This FeatherWing will make it real easy to add datalogging to any of our existing Feathers. You get both an I2C real time clock (PCF8523) with 32KHz crystal and battery backup, and a microSD socket that connects to the SPI port pins (+ extra pin for CS). Tested and works great with any of our Feathers, based on ATmega32u4, ATSAMD21, Teensy, or ESP8266.

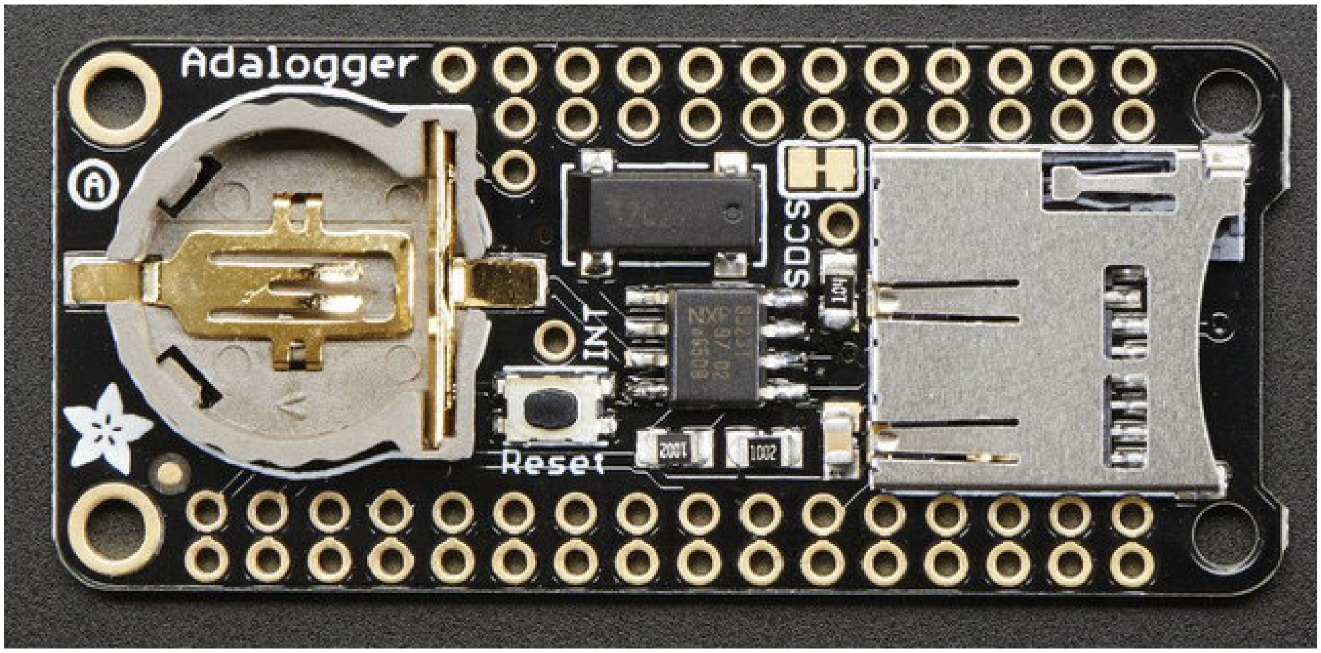


We recommend the Arduino's default SD library to talk to the microSD card socket. On ESP8266, the SD CS pin is on GPIO 15, on Atmel M0 or 32u4 it's on GPIO 10. You can cut the trace to the default pin and change this to any pin. To use the RTC, use our RTClib library (http://adafru.it/c7r). If you need a precision RTC, check out our DS3231 FeatherWing (http://adafru.it/3028)



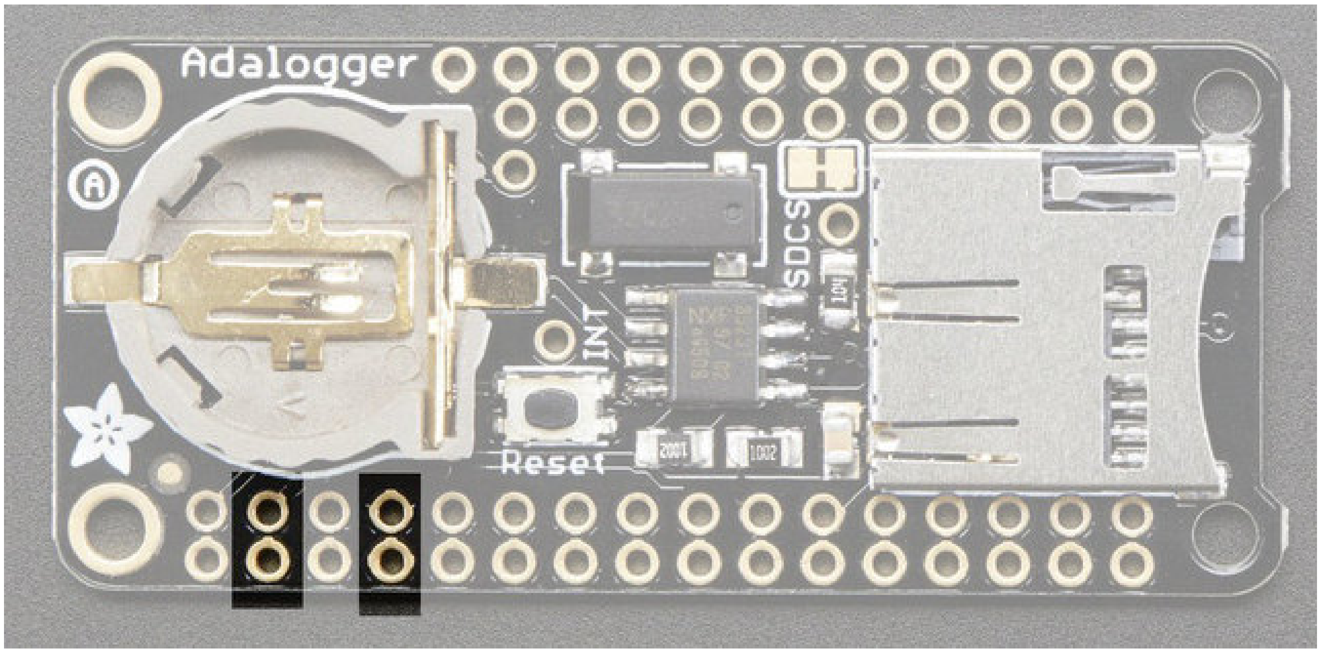
Great for any kind of datalogging or even data *reading*! Some light soldering is required to attach the headers onto the 'Wing but it's a 10 minute task.

Pinouts



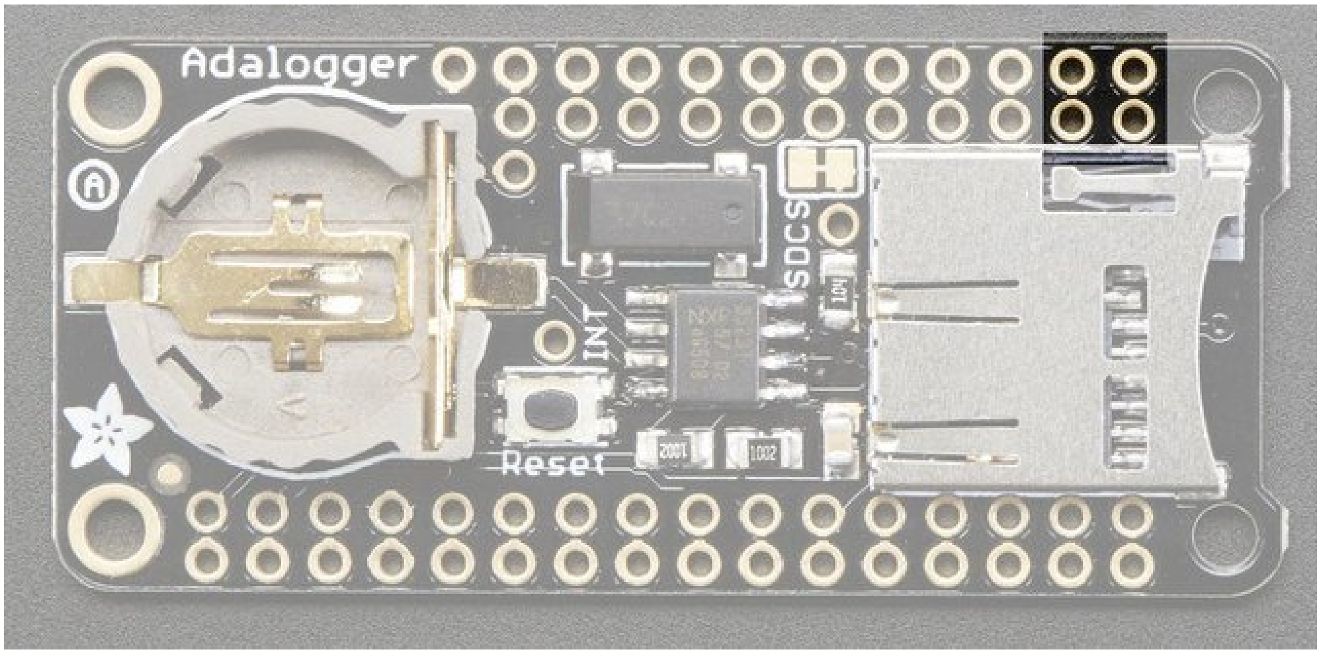
Evern though every pin from the Feather is 'doubled up' with an inner header, not all of the pins are actually used!

Power Pins



On the bottom row, the **3.3V** (leftmost) and **GND** (third from left) pin are used to power the SD card and RTC (to take a load off the coin cell battery when main power is available)

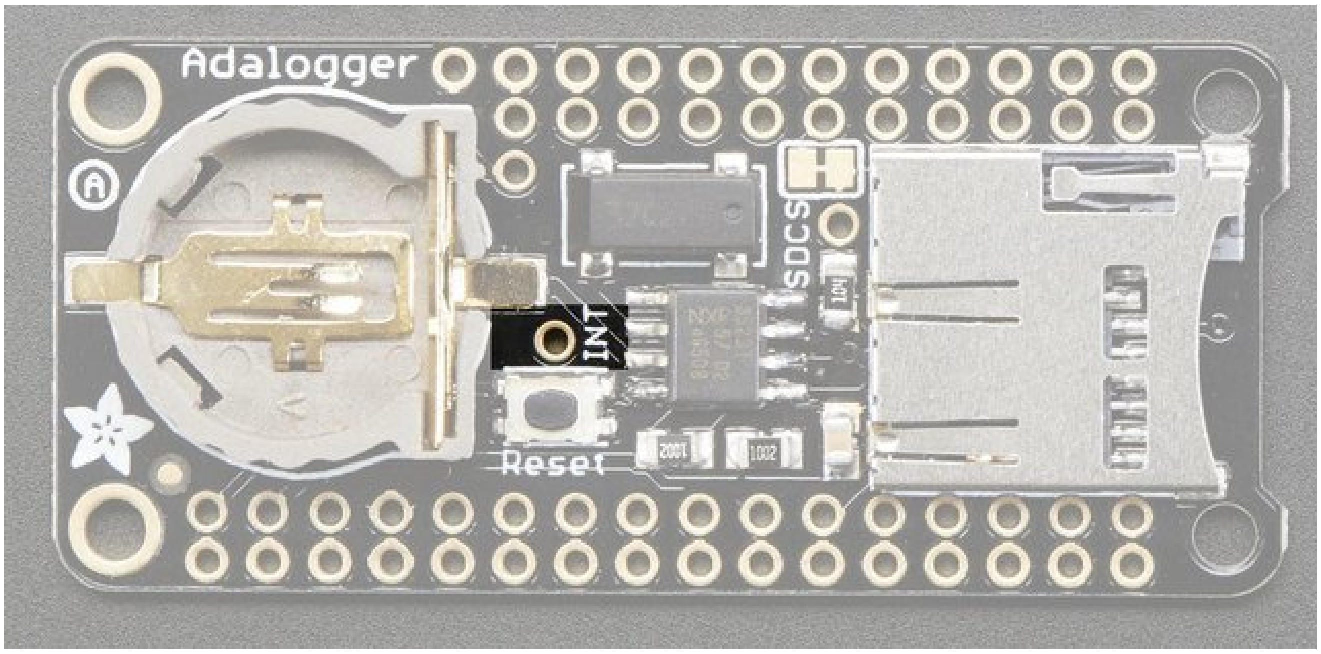
RTC & I2C Pins



In the top right, **SDA** (rightmost) and **SCL** (to the left of SDA) are used to talk to the RTC chip.

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. This pin has a 10K pullup resistor to 3.3V
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. This pin has a 10K pullup resistor to 3.3V

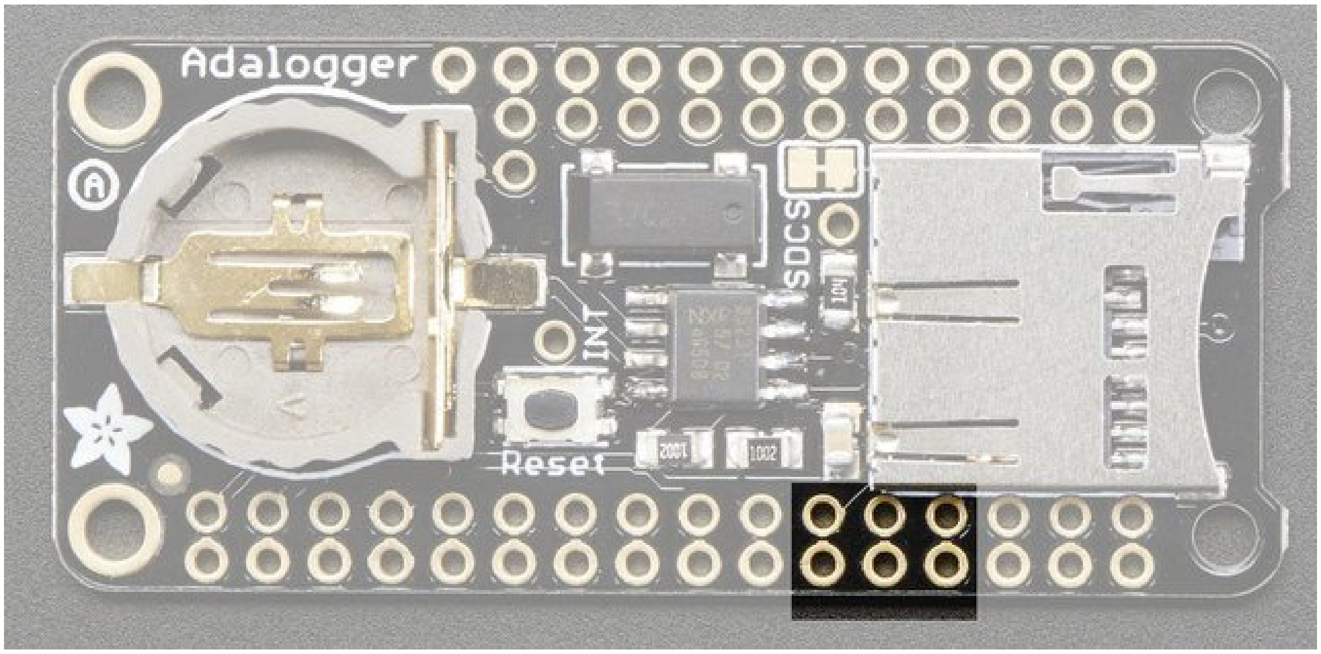
These pins are in the same location on every Feather



There's also a breakout for **INT** which is the output pin from the RTC. It can be used as an interrupt output or it could also be used to generate a square wave.

Note that this pin is **open drain** - you must enable the internal pullup on whatever digital pin it is connected to!

SD & SPI Pins



Starting from the left you've got

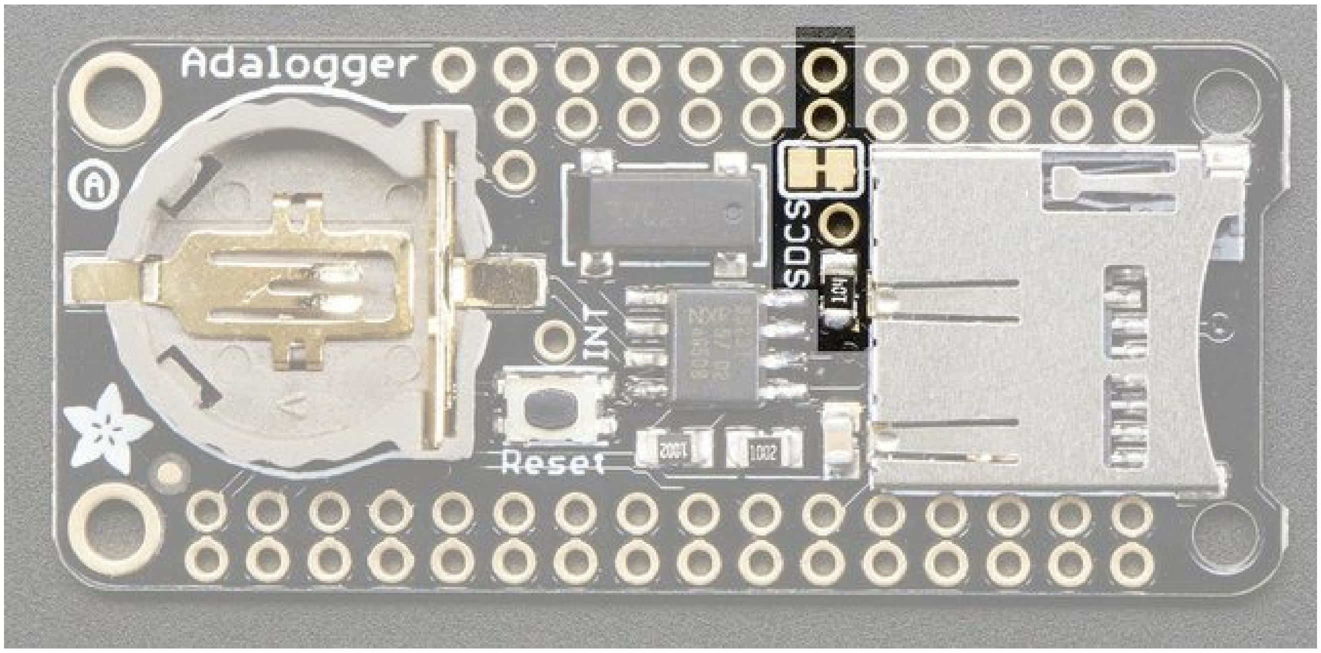
- **SPI Clock** (SCK) - output from feather to wing
- **SPI Master Out Slave In** (MOSI) - output from feather to wing
- **SPI Master In Slave Out** (MISO) - input from wing to feather

These pins are in the same location on every Feather. They are used for communicating with the SD card. When the SD card is not inserted, these pins are completely free. MISO is tri-stated whenever the **SD CS** pin is pulled high

The **SDCS** pin is the chip select line.

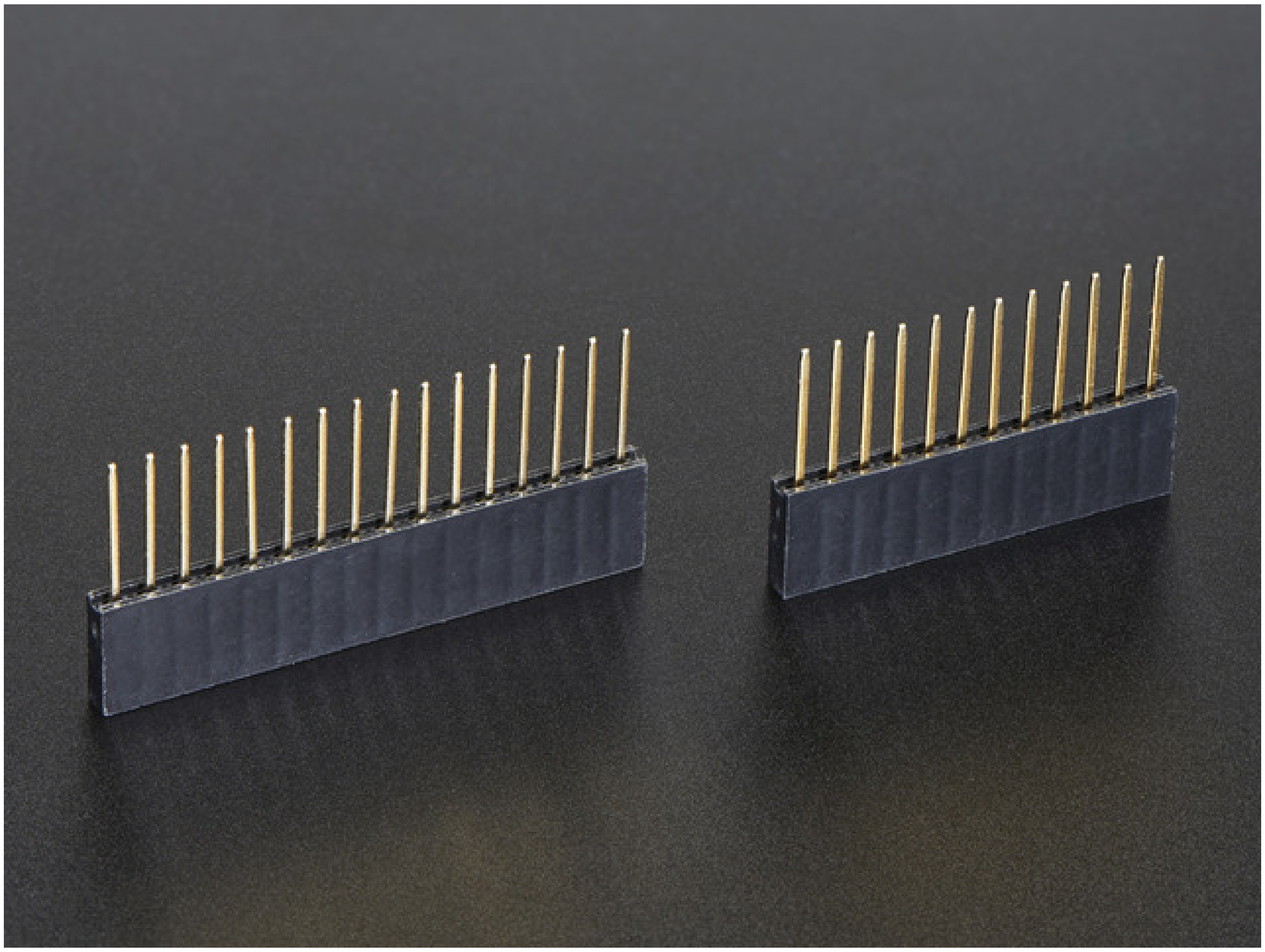
- On ESP8266, the SD CS pin is on GPIO 15
- On Atmel M0 or 32u4 it's on GPIO 10
- On Teensy 3.x it's on GPIO 10

You can cut the trace to the default pin and change this to any pin by soldering a wire to any available pad.

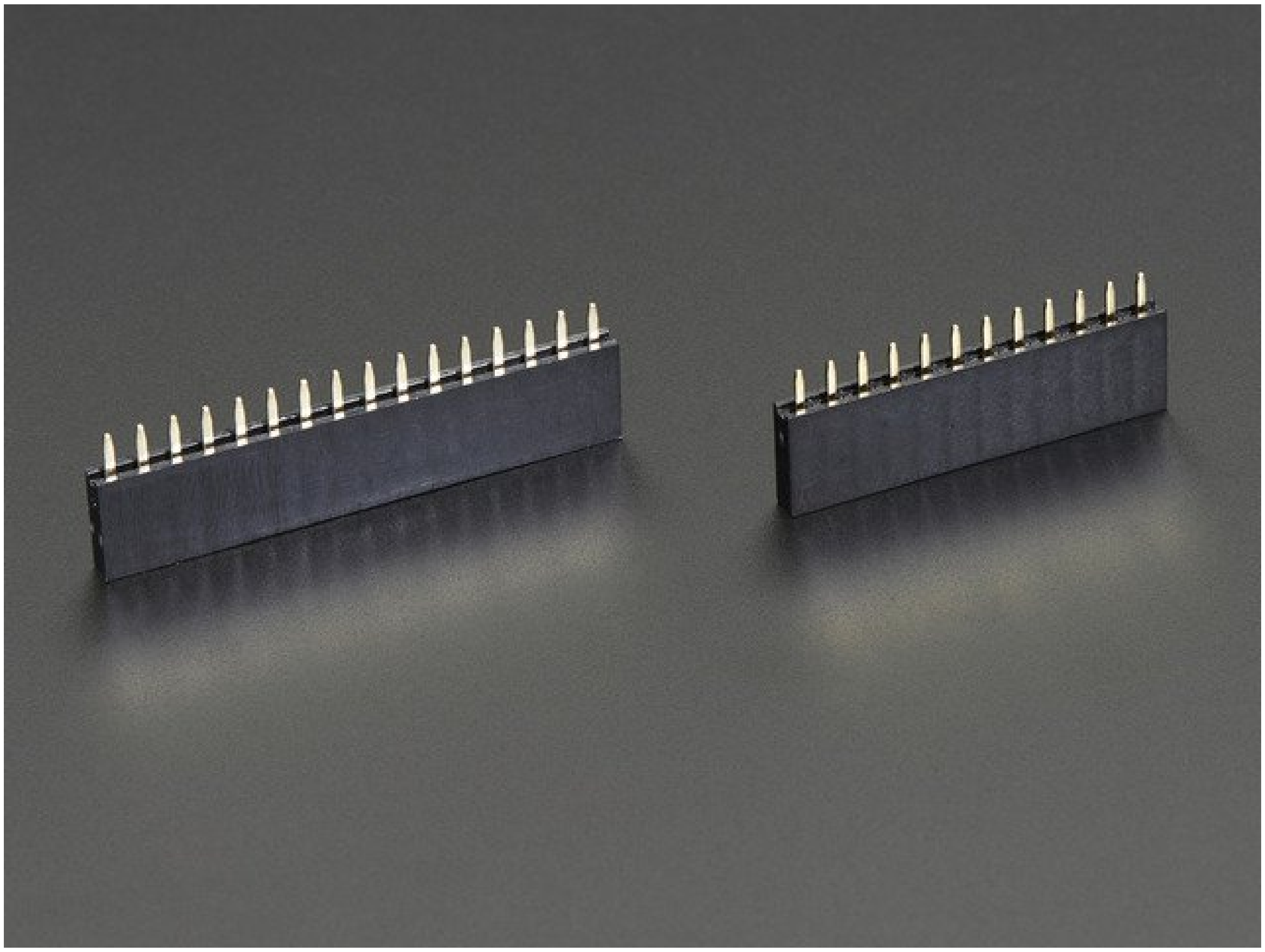


Assembly

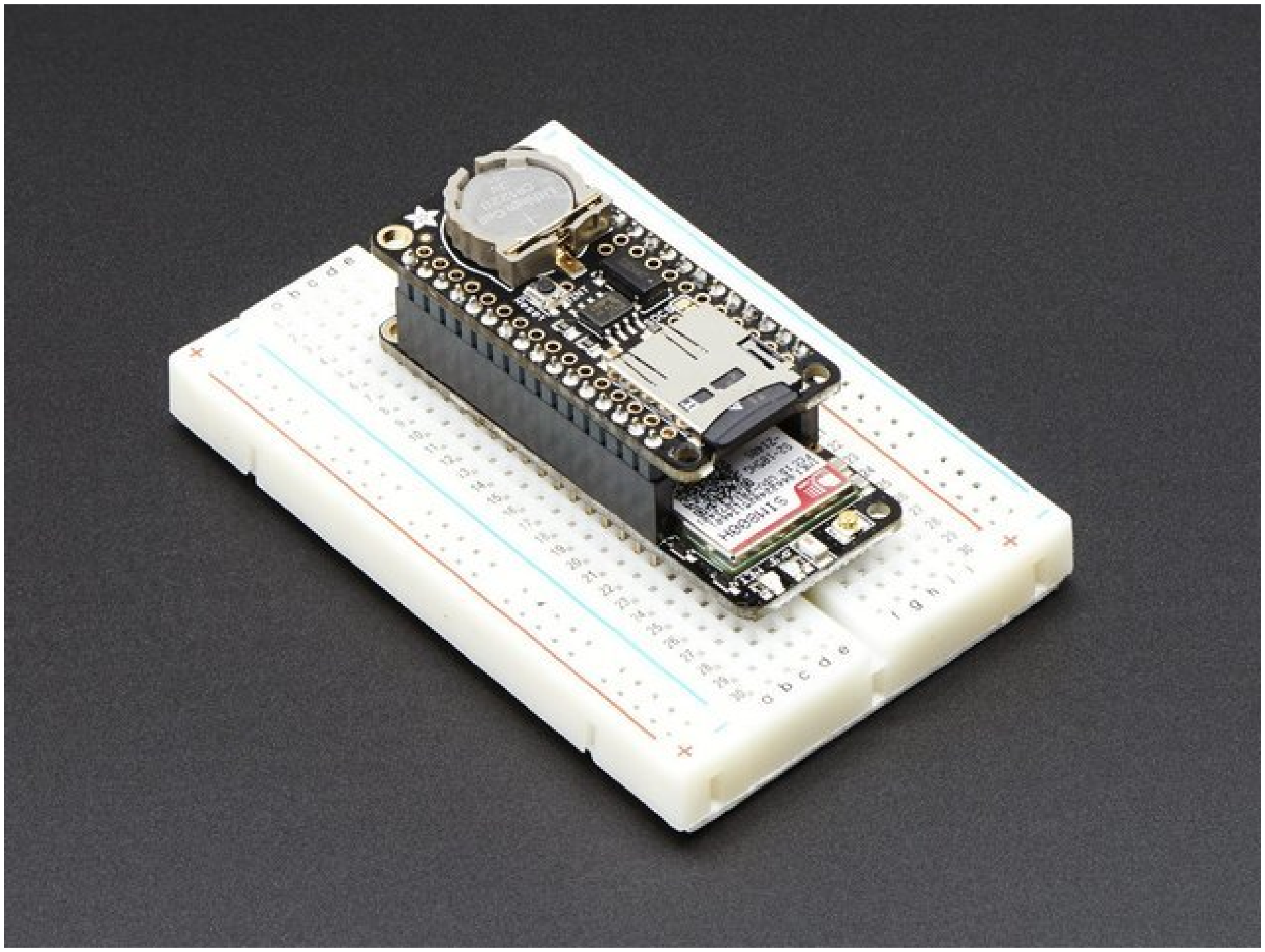
When putting together your Featherwings, think about how you want it to connect, you can use stacking headers:



Or plain female socket headers:



The most common method of attachment for the featherwing is putting stacking or female headers on the *Feather mainboard* and then putting the Wing on top:



But don't forget, you can **also put the stacking headers on the wing and stack the Feather on top of it!**

Using the Real Time Clock

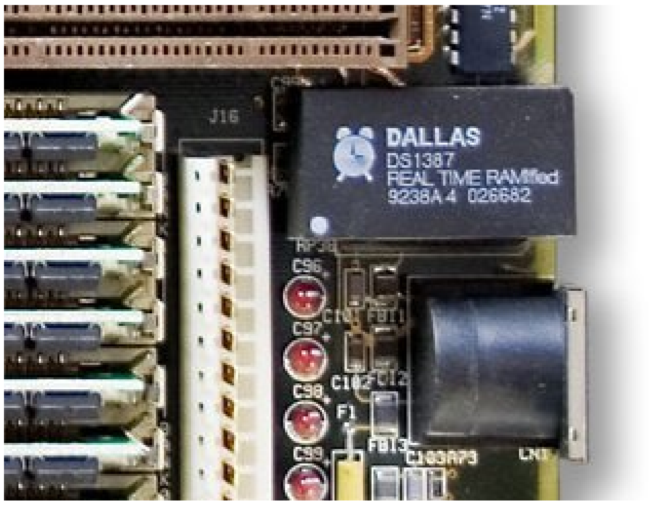
What is a Real Time Clock?

When logging data, it's often really really useful to have timestamps! That way you can take data one minute apart (by checking the clock) or noting at what time of day the data was logged.

The Arduino IDE does have a built-in timekeeper called **millis()** and theres also timers built into the chip that can keep track of longer time periods like minutes or days. So why would you want to have a separate RTC chip? Well, the biggest reason is that **millis()** only keeps track of time *since the Feather was last powered* **-** that means that when the power is turned on, the millisecond timer is set back to 0. The Feather doesnt know its 'Tuesday' or 'March 8th' all it can tell is 'Its been 14,000 milliseconds since I was last turned on'.

OK so what if you wanted to set the time? You'd have to program in the date and time and you could have it count from that point on. But if it lost power, you'd have to reset the time. Much like very cheap alarm clocks: every time they lose power they blink **12:00**

While this sort of basic timekeeping is OK for some projects, a data-logger will need to have **consistent timekeeping that doesnt reset when the power goes out or is reprogrammed**. Thus, we include a separate RTC! The RTC chip is a specialized chip that just keeps track of time. It can count leap-years and knows how many days are in a month, but it doesn't take care of Daylight Savings Time (because it changes from place to place)



This image shows a computer motherboard with a Real Time Clock called the DS1387 (http://adafru.it/aX0). Theres a lithium battery in there which is why it's so big.

The RTC we'll be using is the *PCF8523* *(http://adafru.it/reb)*

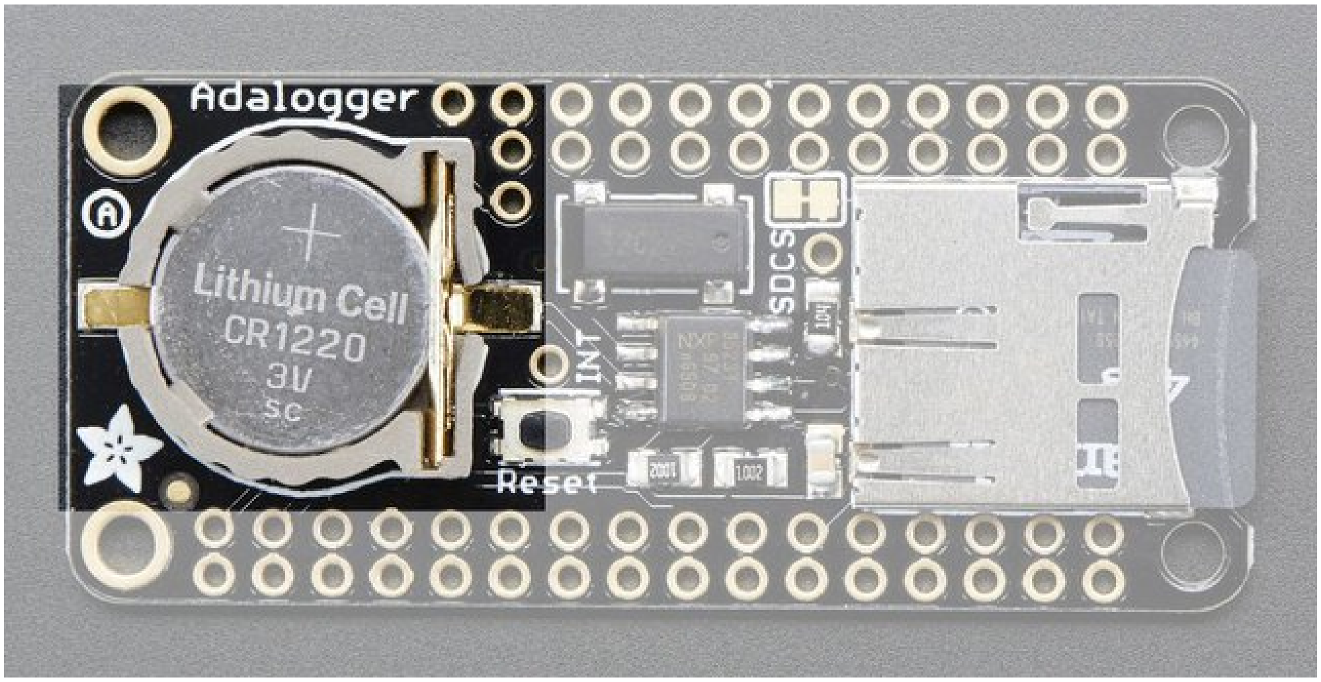
Battery Backup

As long as it has a coin cell to run it, the RTC will merrily tick along for a long time, even when the Feather loses power, or is reprogrammed.

Use any CR1220 3V lithium metal coin cell battery:

CR1220 12mm Diameter - 3V Lithium Coin Cell Battery

PRODUCT ID: 380
These are the highest quality & capacity batteries, the same as shipped with the iCufflinks, iNecklace, Datalogging and GPS Shields, GPS HAT, etc. One battery per order...
http://adafru.it/em8
$0.95
IN STOCK
You MUST have a coin cell installed for the RTC to work, if there is no coin cell, it will act strangely and possibly hang the Arduino when you try to use it, so ALWAYS make SURE there's a battery installed, even if it's a dead battery.



Talking to the RTC

The RTC is an i2c device, which means it uses 2 wires to to communicate. These two wires are used to set the time and retrieve it.

For the RTC library, we'll be using a fork of JeeLab's excellent RTC library, which is available on GitHub (http://adafru.it/c7r). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download RTC Library
http://adafru.it/cxm

Rename the uncompressed folder **RTClib** and check that the **RTClib** folder contains **RTClib.cpp** and **RTClib.h**

Place the **RTClib** library folder your **arduinosketchfolder**/**libraries**/ folder.
You may need to create the **libraries** subfolder if it's your first library. Restart the IDE.

We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use

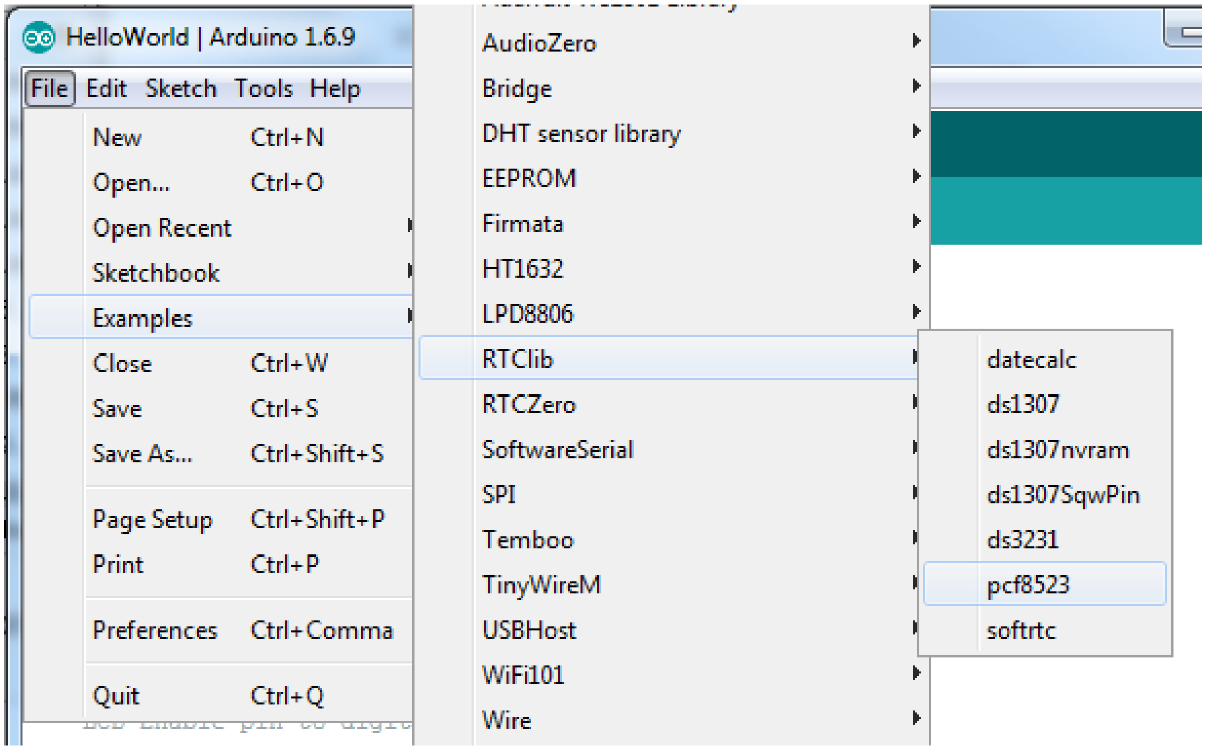
Once done, restart the IDE

First RTC test

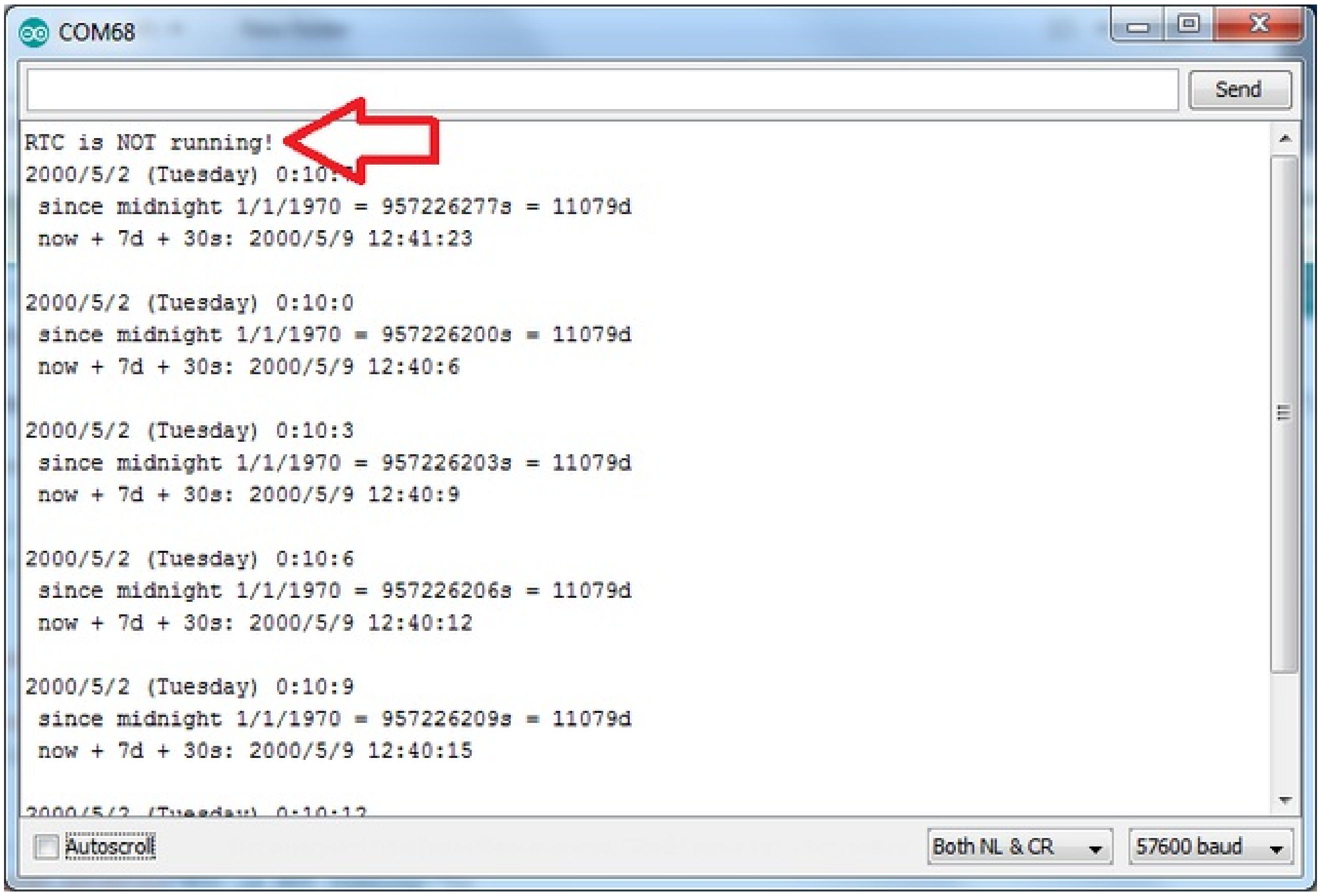
The first thing we'll demonstrate is a test sketch that will read the time from the RTC once a second. We'll also show what happens if you remove the battery and replace it since that causes the RTC to halt. So to start, remove the battery from the holder while the Feather is not powered or plugged into USB. Wait 3 seconds and then replace the battery. This resets the RTC chip. Now load up the matching sketch for your RTC

Open up **Examples->RTClib->pcf8523**

Upload it to your Feather with the Wing on!



Now open up the Serial Console and make sure the baud rate is set correctly at**57600 baud** you should see the following:



```
RTC is NOT running!
2000/5/2 (Tuesday) 0:10:
 since midnight 1/1/1970 = 957226277s = 11079d
 now + 7d + 30s: 2000/5/9 12:41:23

2000/5/2 (Tuesday) 0:10:0
 since midnight 1/1/1970 = 957226200s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:6

2000/5/2 (Tuesday) 0:10:3
 since midnight 1/1/1970 = 957226203s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:9

2000/5/2 (Tuesday) 0:10:6
 since midnight 1/1/1970 = 957226206s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:12

2000/5/2 (Tuesday) 0:10:9
 since midnight 1/1/1970 = 957226209s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:15

2000/5/2 (Tuesday) 0:10:12
```

Whenever the RTC chip loses all power (including the backup battery) it will reset to an earlier date and report the time as 0:0:0 or similar. Whenever you set the time, this will kickstart the clock ticking.

So, basically, the upshot here is that you should never ever remove the battery once you've set the time. You shouldn't have to and the battery holder is very snug so unless the board is crushed, the battery won't 'fall out'

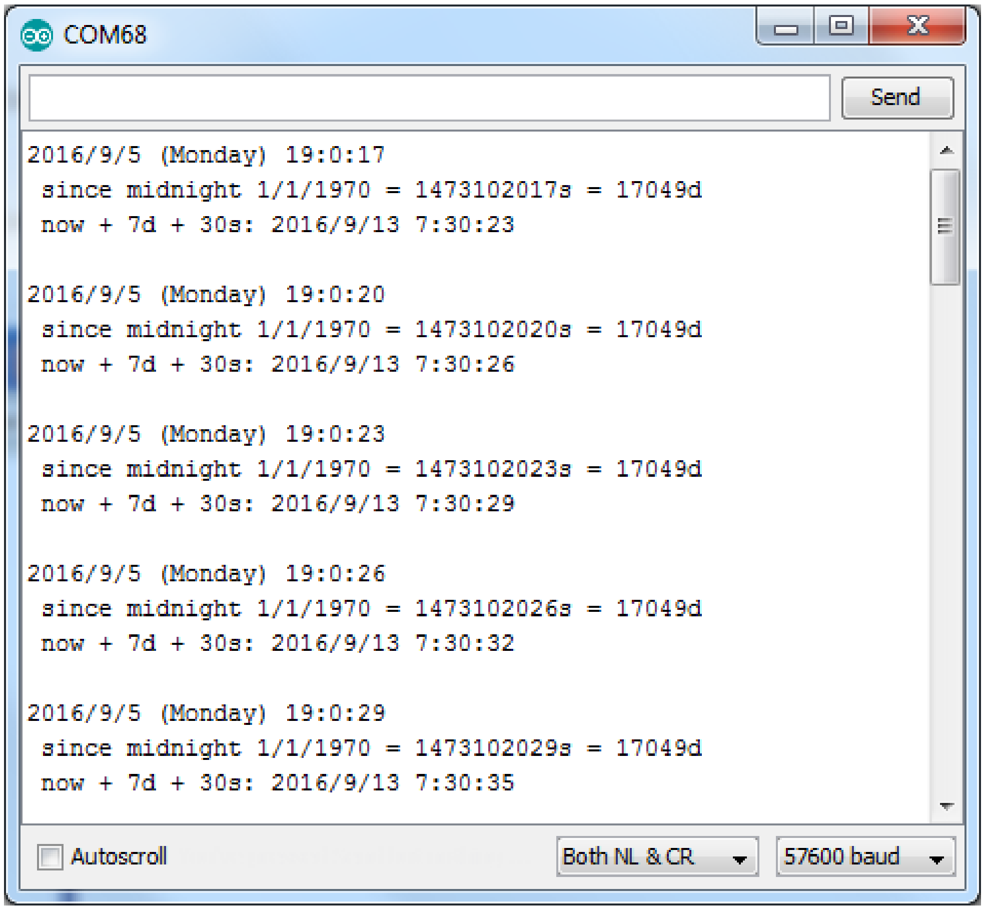
Setting the time

With the same sketch loaded, uncomment the line that starts with**RTC.adjust** like so:

```
if (! rtc.initialized()) {
  Serial.println("RTC is NOT running!");
  // following line sets the RTC to the date & time this sketch was compiled
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
```

This line is very cute, what it does is take the Date and Time according the computer you're using (right when you compile the code) and uses that to program the RTC. If your computer time is not set right you should fix that first. Then you must press the **Upload** button to compile and then immediately upload. If you compile and then upload later, the clock will be off by that amount of time.

Then open up the Serial monitor window to show that the time has been set



```
2016/9/5 (Monday) 19:0:17
 since midnight 1/1/1970 = 1473102017s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:23

2016/9/5 (Monday) 19:0:20
 since midnight 1/1/1970 = 1473102020s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:26

2016/9/5 (Monday) 19:0:23
 since midnight 1/1/1970 = 1473102023s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:29

2016/9/5 (Monday) 19:0:26
 since midnight 1/1/1970 = 1473102026s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:32

2016/9/5 (Monday) 19:0:29
 since midnight 1/1/1970 = 1473102029s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:35
```

From now on, you won't have to ever set the time again: the battery will last 5 or more years

Reading the time

Now that the RTC is merrily ticking away, we'll want to query it for the time. Let's look at the sketch again to see how this is done

```
void loop () {
  DateTime now = rtc.now();

  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(" (");
  Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
  Serial.print(") ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.println();
```

There's pretty much only one way to get the time using the RTClib, which is to call **now()**, a function that returns a DateTime object that describes the year, month, day, hour, minute and second when you called **now()**.

There are some RTC libraries that instead have you call something like **RTC.year()** and **RTC.hour()** to get the current year and hour. However, there's one problem where if you happen to ask for the minute right at **3:14:59** just before the next minute rolls over, and then the second right after the minute rolls over (so at **3:15:00**) you'll see the time as **3:14:00** which is a minute off. If you did it the other way around you could get **3:15:59** - so one minute off in the other direction.

Because this is not an especially unlikely occurance - particularly if you're querying the time pretty often - we take a 'snapshot' of the time from the RTC all at once and then we can pull it apart into **day()** or **second()** as seen above. It's a tiny bit more effort but we think its worth it to avoid mistakes!

We can also get a 'timestamp' out of the DateTime object by calling **unixtime** which counts the number of seconds (not counting leapseconds) since midnight, January 1st 1970

```
    Serial.print(" since 2000 = ");
    Serial.print(now.unixtime());
    Serial.print("s = ");
    Serial.print(now.unixtime() / 86400L);
    Serial.println("d");
```

Since there are 60*60*24 = 86400 seconds in a day, we can easily count days since then as well. This might be useful when you want to keep track of how much time has passed since the last query, making some math a lot easier (like checking if it's been 5 minutes later, just see if **unixtime()** has increased by 300, you dont have to worry about hour changes)

Using the SD Card

The other half of the adalogger FeatherWing is the SD card. The SD card is how we store long term data. While the Feather may have a permanent EEPROM storage, its only a couple hundred bytes - tiny compared to a 2 gig SD card. SD cards are so cheap and easy to get, its an obvious choice for long term storage so we use them for the 'Wing!

The FeatherWing kit doesn't come with an SD card butwe carry one in the shop that is guaranteed to work (http://adafru.it/aIH). Pretty much any SD card should work but be aware that some cheap cards are 'fakes' and can cause headaches.

4GB Blank SD/MicroSD Memory Card

PRODUCT ID: 102
Add mega-storage in a jiffy using this 4 GB micro-SD card. It comes with a SD adapter so you can use it with any of our shields or adapters! Preformatted to FAT so it works out of the box...
http://adafru.it/eZR
$7.95
IN STOCK

You'll also need a way to read and write from the SD card. Sometimes you can use your camera and MP3 player - when its plugged in you will be able to see it as a disk. Or you may need an SD card reader (http://adafru.it/939). The Wing **doesnt** have the ability to display the SD card as a 'hard disk' like some MP3 players or games, the Feather does not have the hardware for that, so you will need an external reader!

USB MicroSD Card Reader/Writer - microSD / microSDHC / microSDXC

PRODUCT ID: 939
This is the cutest little microSD card reader/writer - but don't be fooled by its adorableness! It's wicked fast and supports up to 64 GB SDXC cards! Simply slide the card into the edge...
http://adafru.it/ree
$5.95
IN STOCK

Formatting under Windows/Mac

If you bought an SD card, chances are it's already pre-formatted with a FAT filesystem. However you may have problems with how the factory formats the card, or if it's an old card it needs to be reformatted. The Arduino SD library we use supports both **FAT16** and **FAT32** filesystems. If you have a very small SD card, say 8-32 Megabytes you might find it is formatted **FAT12** which isnt supported. You'll have to reformat these card. Either way, its **always** good idea to format the card before using, even if its new! Note that formatting will erase the card so save anything you want first

We strongly recommend you use the official SD card formatter utility - written by the SD association it solves many problems that come with bad formatting!

The official SD formatter is available from
https://www.sdcard.org/downloads/formatter_4/ (http://adafru.it/cfL)

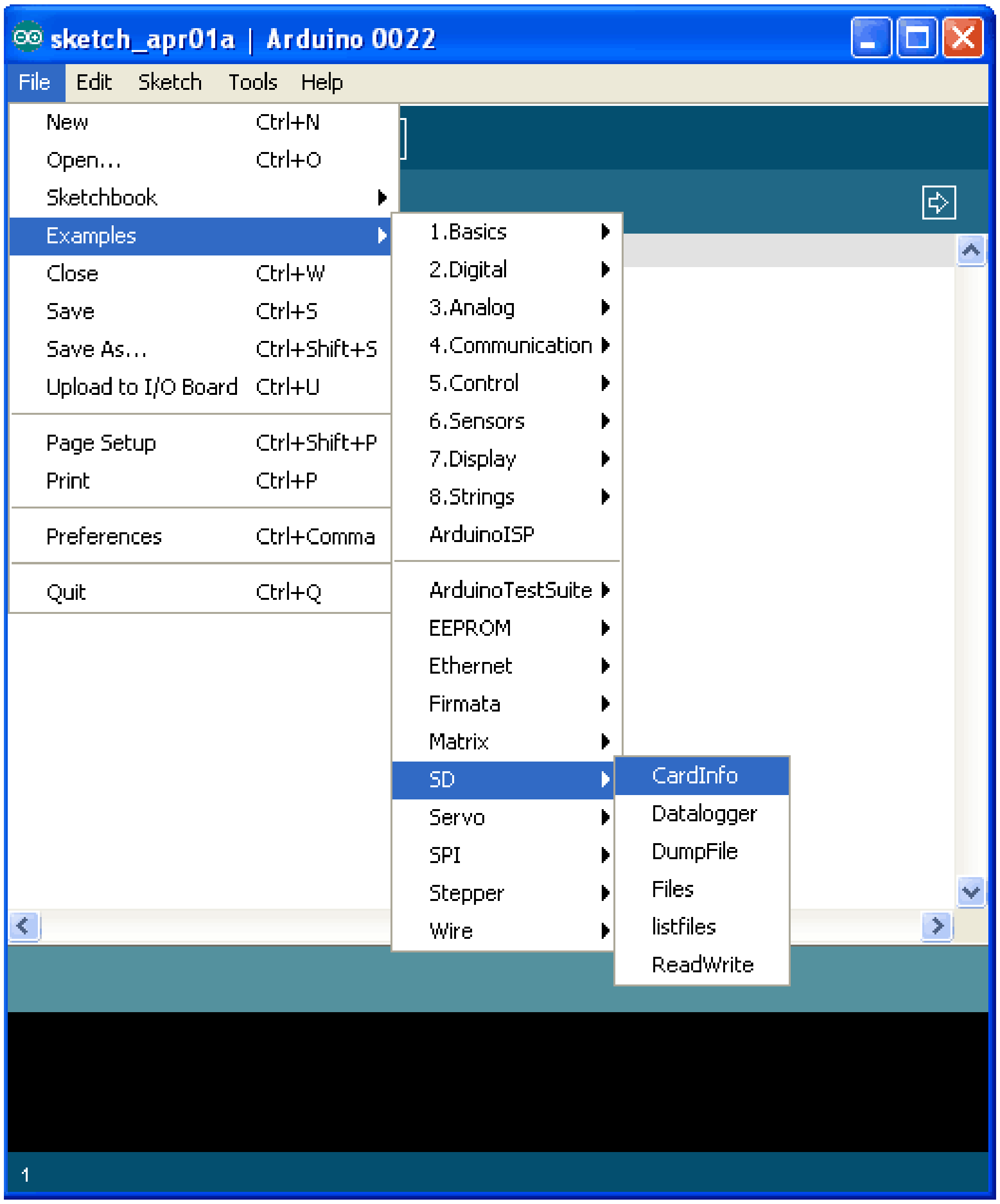
Download it and run it on your computer, there's also a manual linked from that page for use

Download the official SD Formatter software for Windows http://adafru.it/cfL

Get Card Info

The Arduino SD Card library has a built in example that will help you test the Wing and your connections

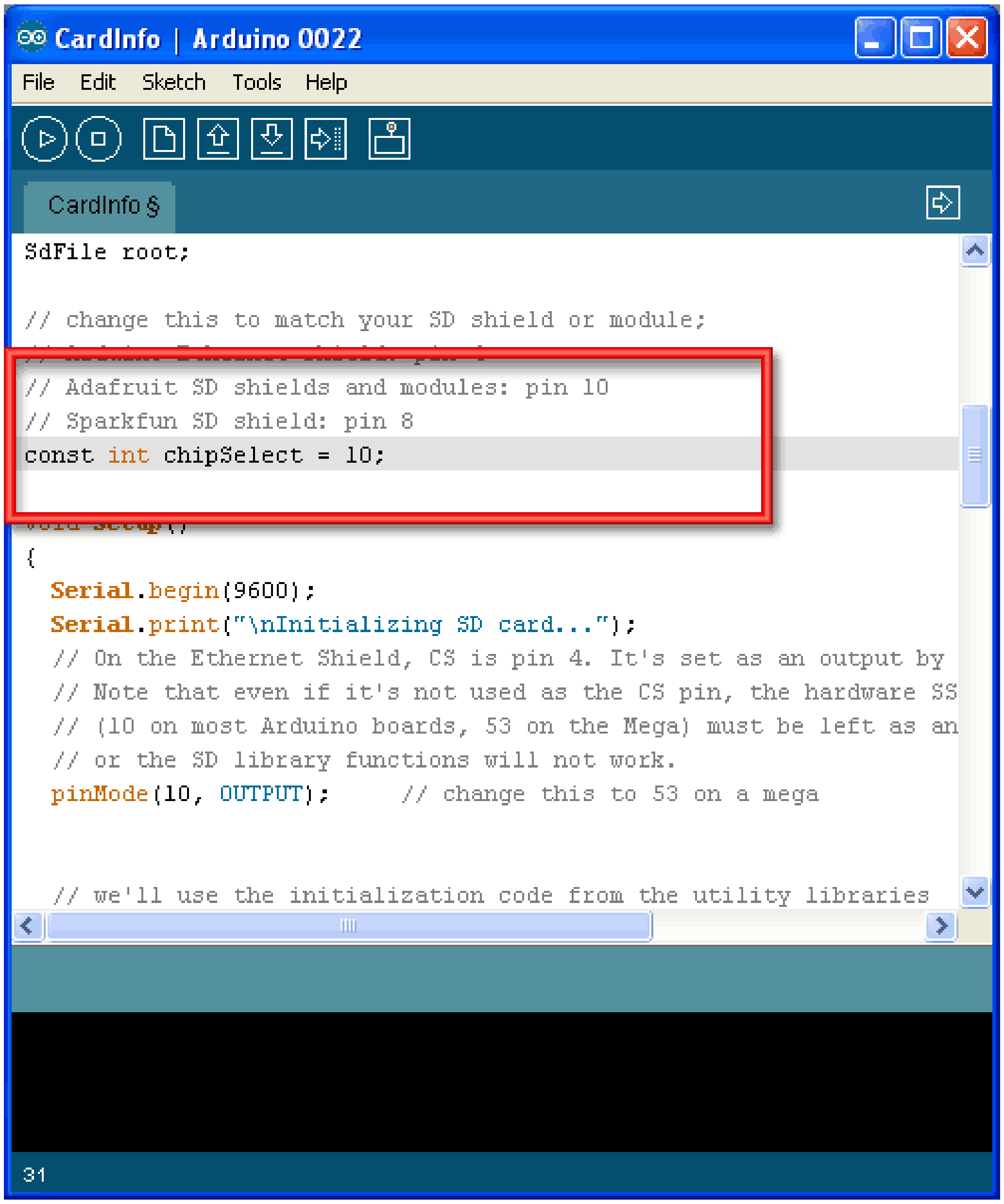
Open the file **CardInfo** example sketch in the **SD** library:



This sketch will not write any data to the card, just tell you if it managed to recognize it, and some information about it. This can be **very** useful when trying to figure out whether an SD card is supported. Before trying out a new card, please try out this sketch!

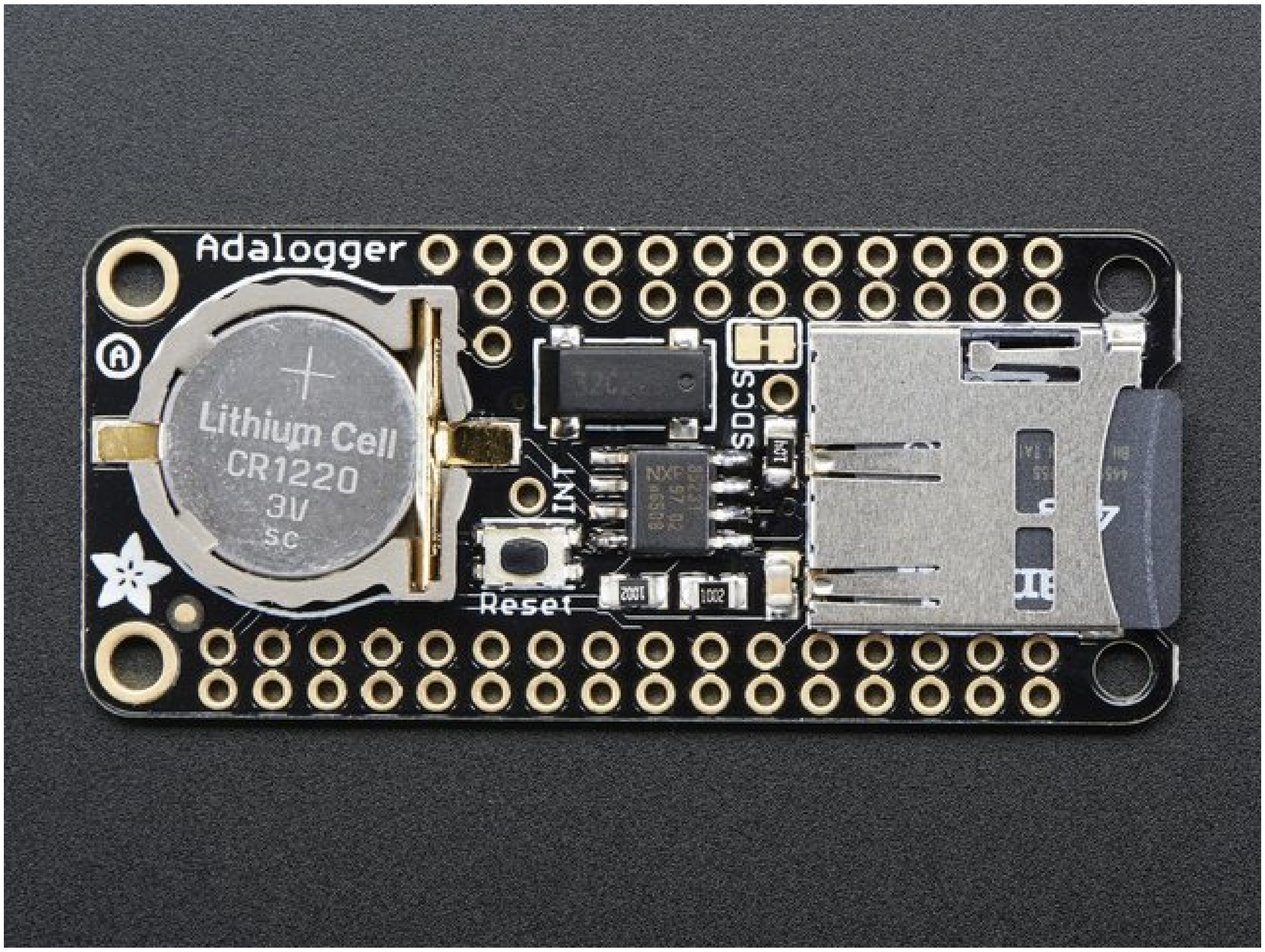
Go to the beginning of the sketch and make sure that the **chipSelect** line is correct.

- On ESP8266, the SD CS pin is on GPIO 15
- On Atmel M0 or 32u4 it's on GPIO 10
- On Teensy 3.x it's on GPIO 10

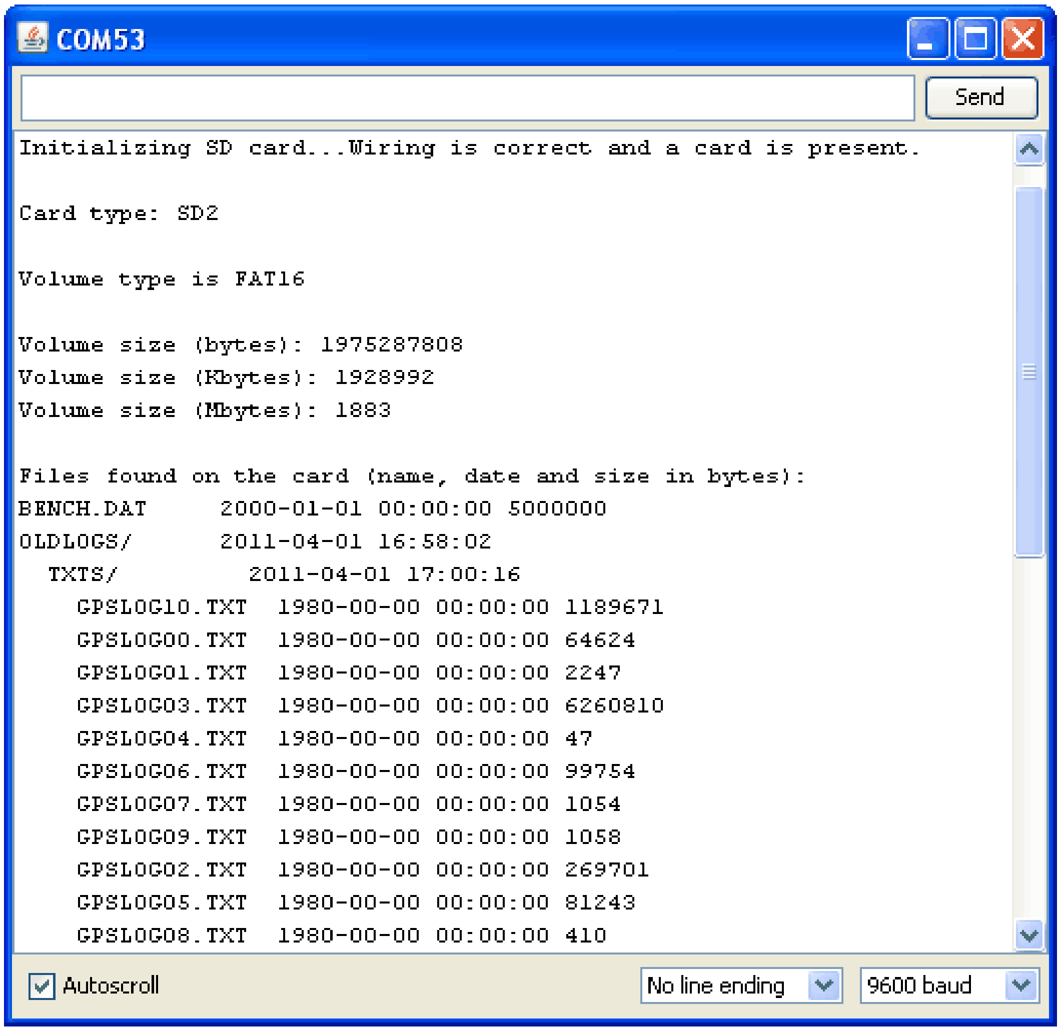


```
SdFile root;

// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
const int chipSelect = 10;

void setup()
{
  Serial.begin(9600);
  Serial.print("\nInitializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by
  // Note that even if it's not used as the CS pin, the hardware SS
  // (10 on most Arduino boards, 53 on the Mega) must be left as an
  // or the SD library functions will not work.
  pinMode(10, OUTPUT);     // change this to 53 on a mega


  // we'll use the initialization code from the utility libraries
```

OK, now insert the micro SD card into the FeatherWing and upload the sketch



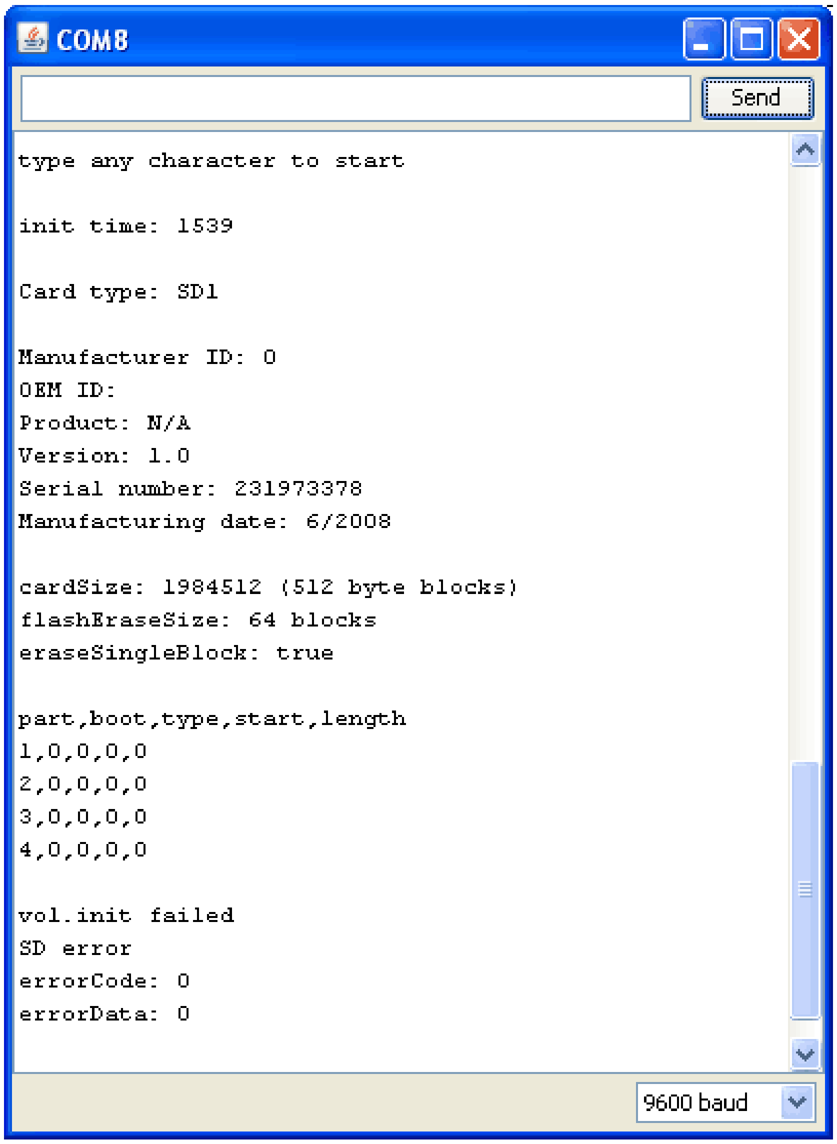
Open up the Serial Monitor and type in a character into the text box (& hit send) when prompted. You'll probably get something like the following:



```
Initializing SD card...Wiring is correct and a card is present.

Card type: SD2

Volume type is FAT16

Volume size (bytes): 1975287808
Volume size (Kbytes): 1928992
Volume size (Mbytes): 1883

Files found on the card (name, date and size in bytes):
BENCH.DAT     2000-01-01 00:00:00 5000000
OLDLOGS/      2011-04-01 16:58:02
  TXTS/         2011-04-01 17:00:16
    GPSLOG10.TXT  1980-00-00 00:00:00 1189671
    GPSLOG00.TXT  1980-00-00 00:00:00 64624
    GPSLOG01.TXT  1980-00-00 00:00:00 2247
    GPSLOG03.TXT  1980-00-00 00:00:00 6260810
    GPSLOG04.TXT  1980-00-00 00:00:00 47
    GPSLOG06.TXT  1980-00-00 00:00:00 99754
    GPSLOG07.TXT  1980-00-00 00:00:00 1054
    GPSLOG09.TXT  1980-00-00 00:00:00 1058
    GPSLOG02.TXT  1980-00-00 00:00:00 269701
    GPSLOG05.TXT  1980-00-00 00:00:00 81243
    GPSLOG08.TXT  1980-00-00 00:00:00 410
```

Its mostly gibberish, but its useful to see the**Volume type is FAT16** part as well as the size of the card (about 2 GB which is what it should be) etc.

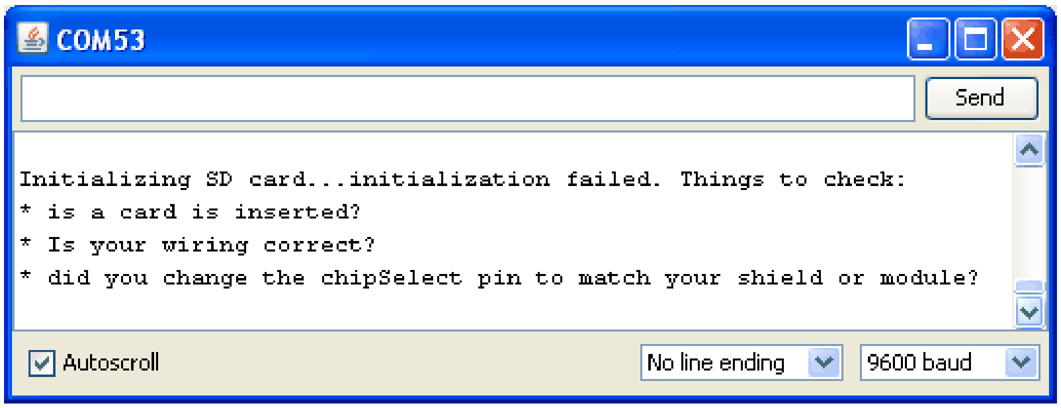
If you have a bad card, which seems to happen more with ripoff version of good brands, you might see:



```
type any character to start

init time: 1539

Card type: SD1

Manufacturer ID: 0
OEM ID:
Product: N/A
Version: 1.0
Serial number: 231973378
Manufacturing date: 6/2008

cardSize: 1984512 (512 byte blocks)
flashEraseSize: 64 blocks
eraseSingleBlock: true

part,boot,type,start,length
1,0,0,0,0
2,0,0,0,0
3,0,0,0,0
4,0,0,0,0

vol.init failed
SD error
errorCode: 0
errorData: 0
```

The card mostly responded, but the data is all bad. Note that the**Product ID** is **"N/A"** and there is no **Manufacturer ID** or **OEM ID**. This card returned some SD errors. Its basically a bad scene, I only keep this card around to use as an example of a bad card! If you get something like this (where there is a response but its corrupted) you should toss the card

Finally, try taking out the SD card and running the sketch again, you'll get the following,



```
Initializing SD card...initialization failed. Things to check:
* is a card is inserted?
* Is your wiring correct?
* did you change the chipSelect pin to match your shield or module?
```

It couldn't even initialize the SD card. This can also happen if there's a soldering error or if the card is *really* damaged

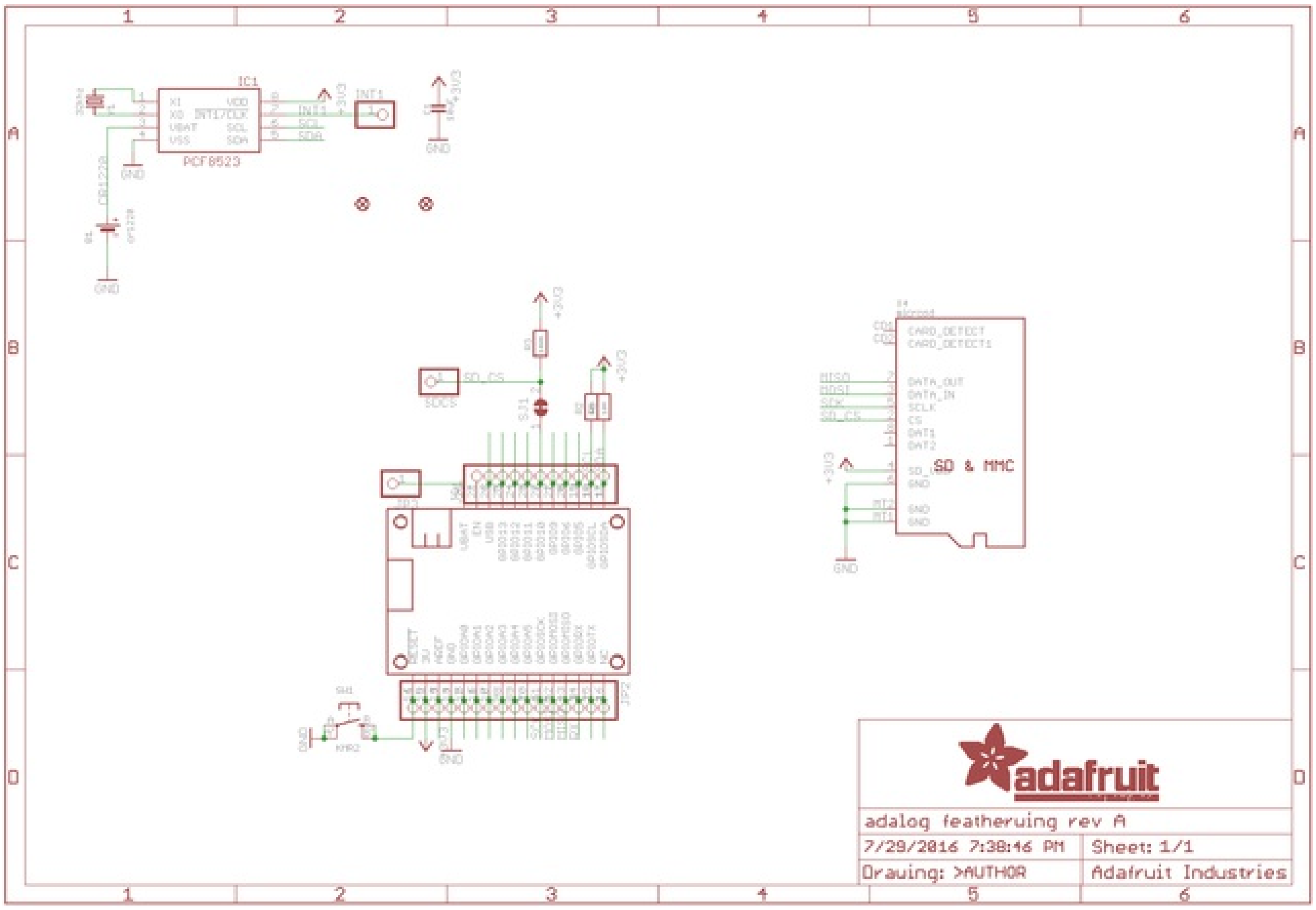
If you're having SD card problems, we suggest using the SD formatter mentioned above first to make sure the card is clean and ready to use!

Downloads

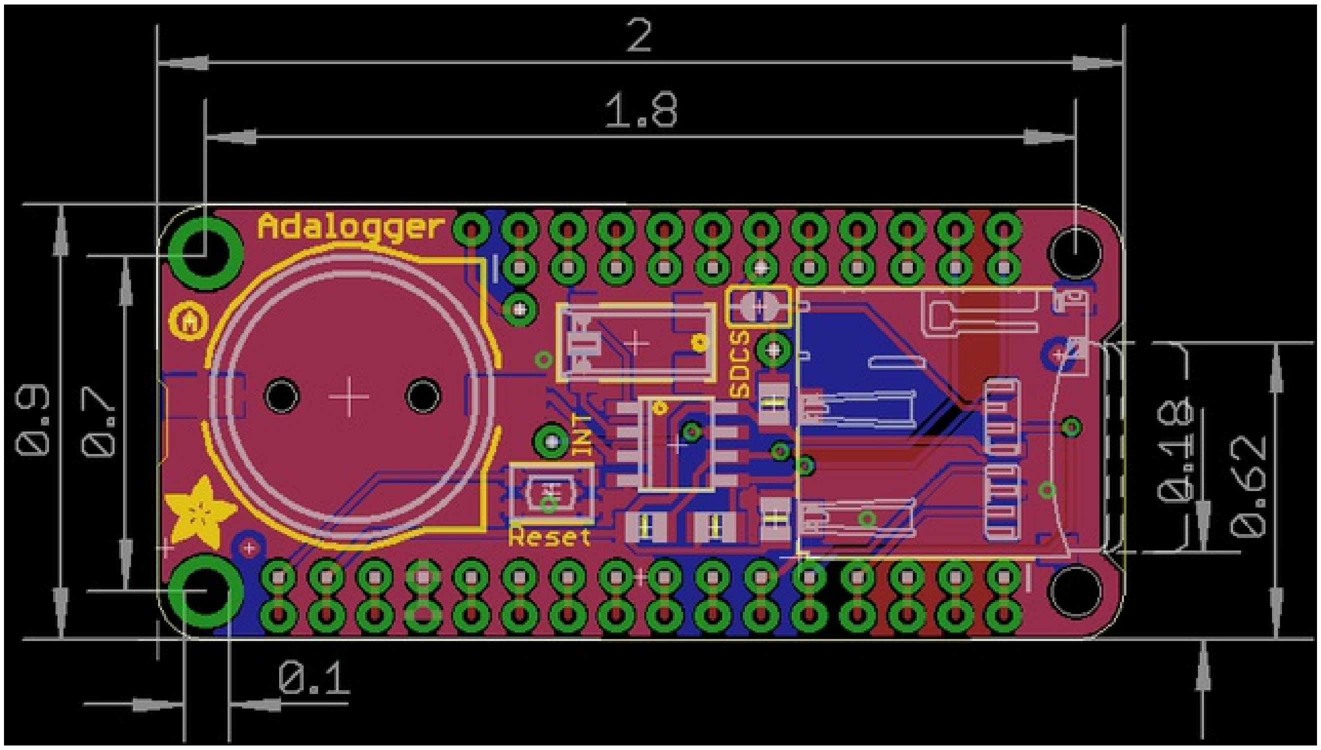
Datasheets and Files

- EagleCAD PCB files on GitHub (http://adafru.it/rek)
- Fritzing object in Adafruit Fritzing library (http://adafru.it/c7M)
- PCF8523 product page (http://adafru.it/reb)

Schematic



Fabrication Print


2
1.8
Adalogger
SDCS
INT
Reset
0.9
0.7
0.18
0.62
0.1