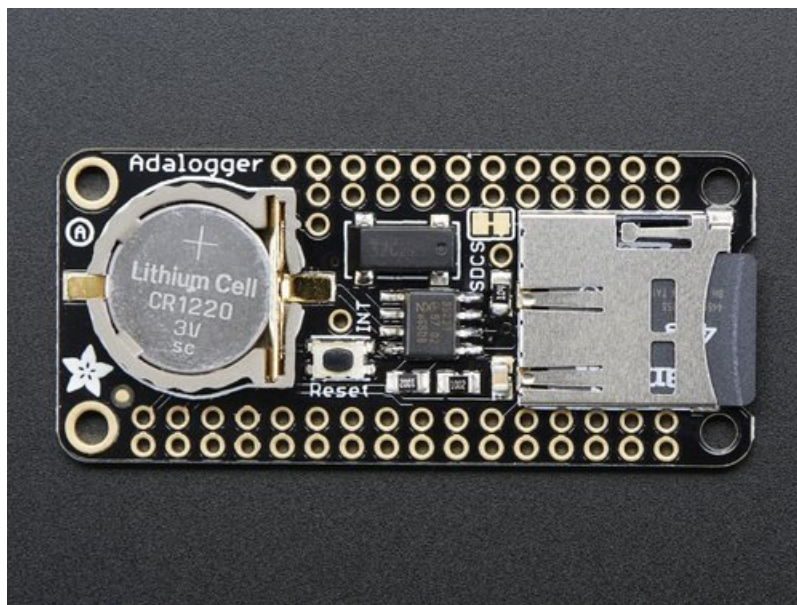


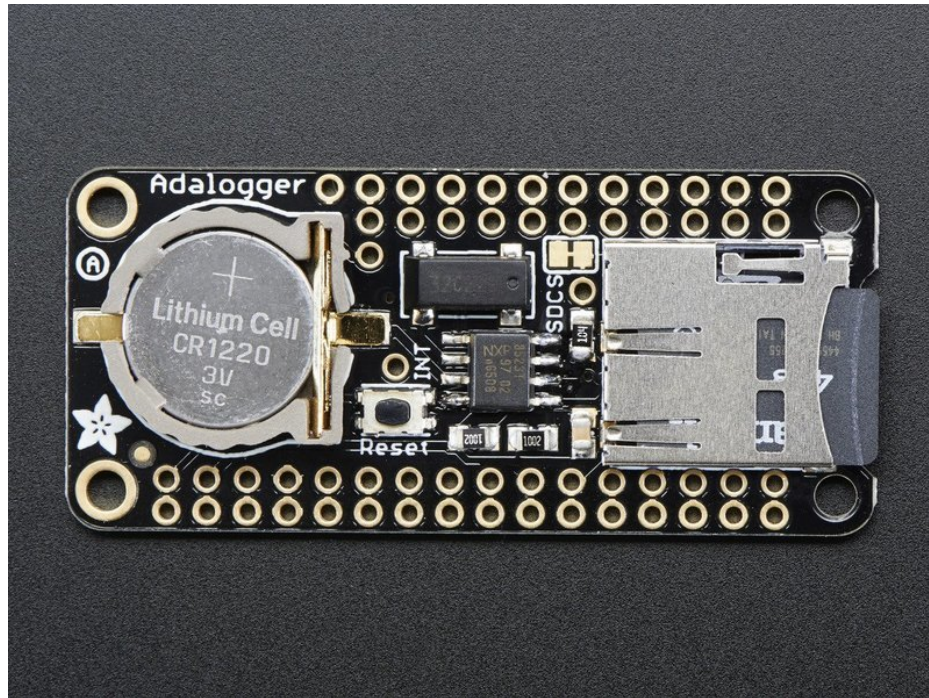
## Adafruit Adalogger FeatherWing

Created by lady ada

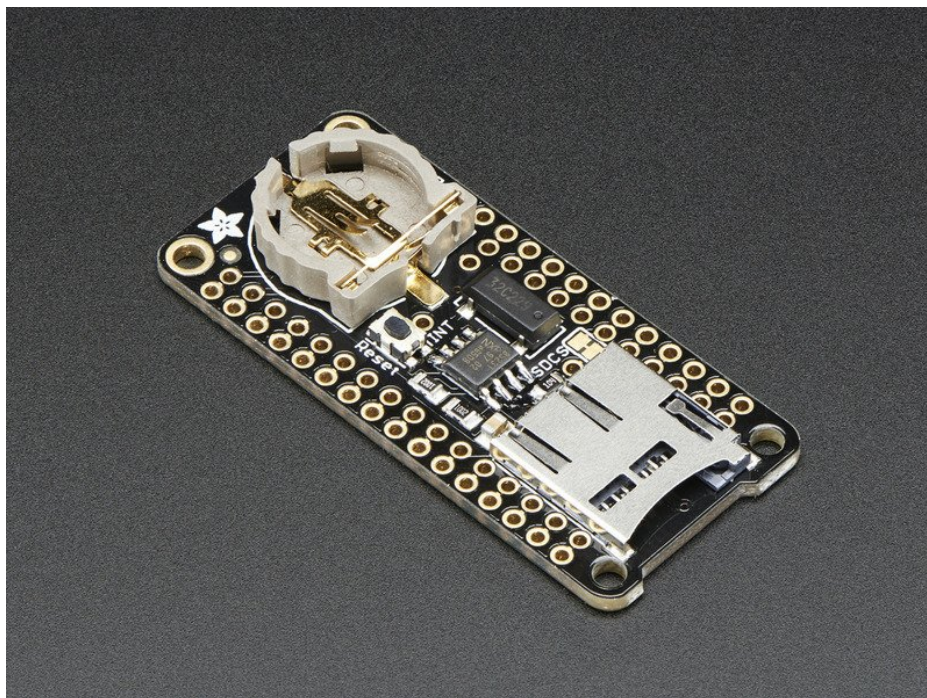


Last updated on 2020-07-28 11:46:17 AM EDT

## Overview



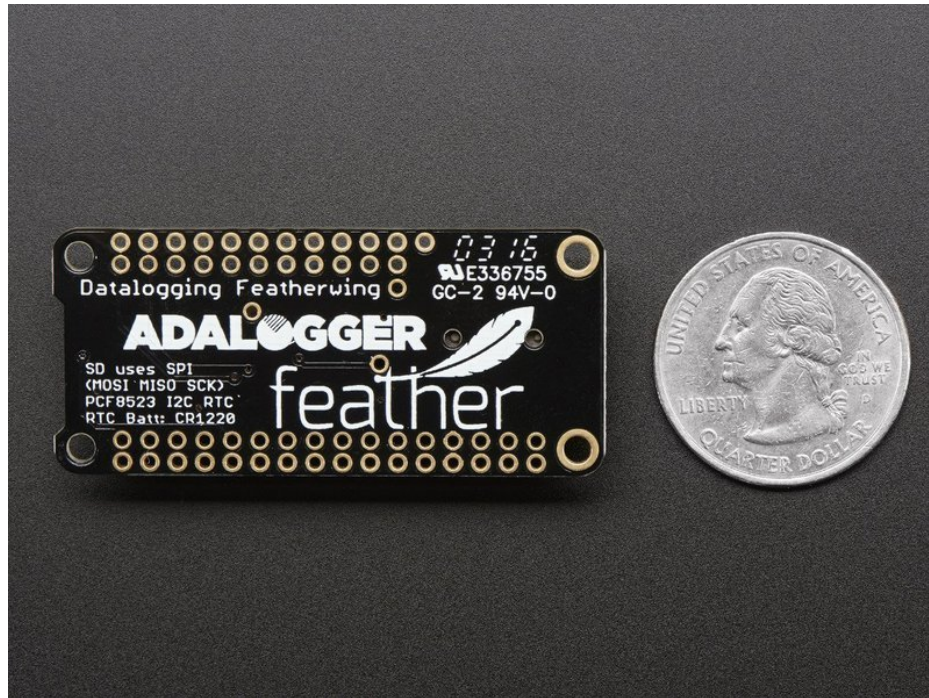
A Feather board without ambition is a Feather board without FeatherWings! This is the **Adalogger FeatherWing**: it adds both a battery-backed Real Time Clock and micro SD card storage to any Feather main board. Using our [Feather Stacking Headers](http://adafru.it/2830) (<http://adafru.it/2830>) or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



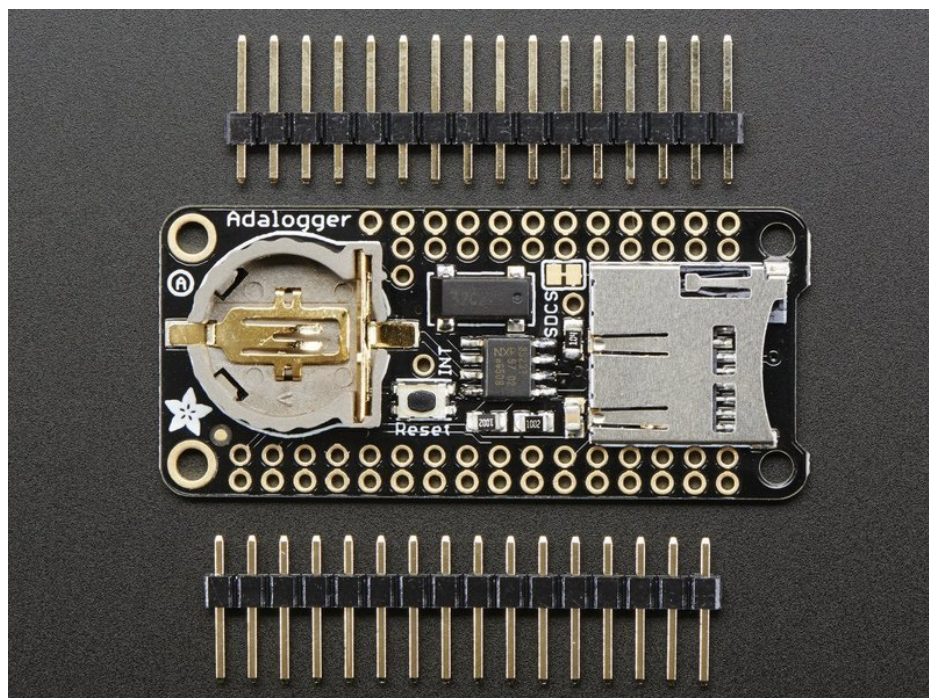
This FeatherWing will make it real easy to add datalogging to any of our existing Feathers. You get both an I2C real time clock (PCF8523) with 32KHz crystal and battery backup, and a microSD socket that connects to the SPI port pins



(+ extra pin for CS). Tested and works great with *any* of our Feathers, based on ATmega32u4, ATmega328P, ATSAMD21, ATSAMD51, nRF52, Teensy, or ESP32/ESP8266.



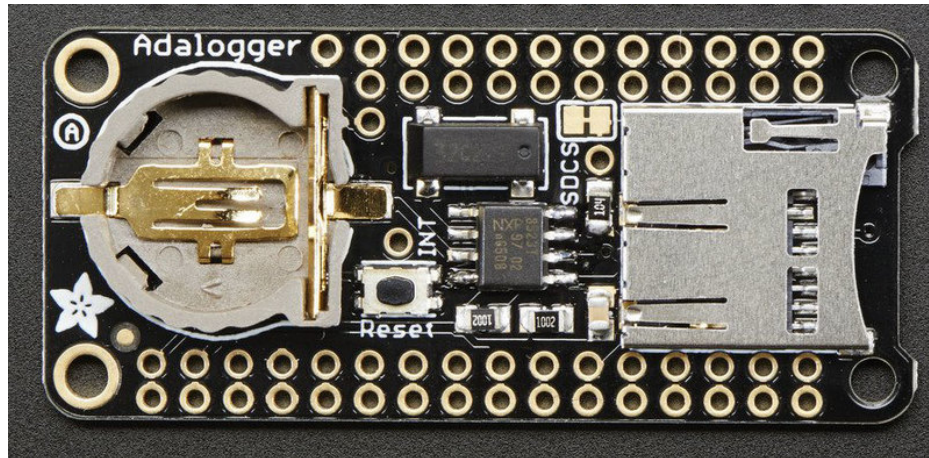
We recommend the Arduino's default SD library to talk to the microSD card socket. On ESP8266, the SD CS pin is on GPIO 15, on Atmel M0, M4, 328P or 32u4 it's on GPIO 10. You can cut the trace to the default pin and change this to any pin. To use the RTC, [use our RTCLib library \(https://adafruit.it/c7r\)](https://adafruit.it/c7r). If you need a [precision RTC, check out our DS3231 FeatherWing \(http://adafruit.it/3028\)](https://adafruit.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.

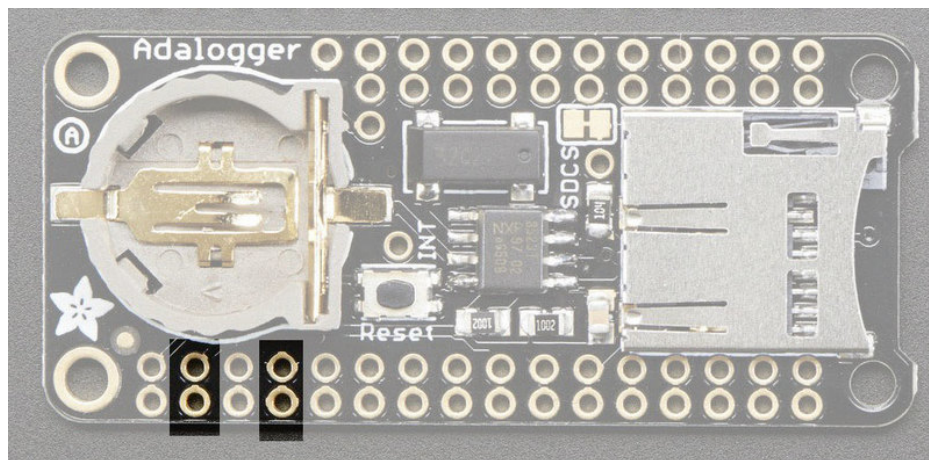


## Pinouts



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

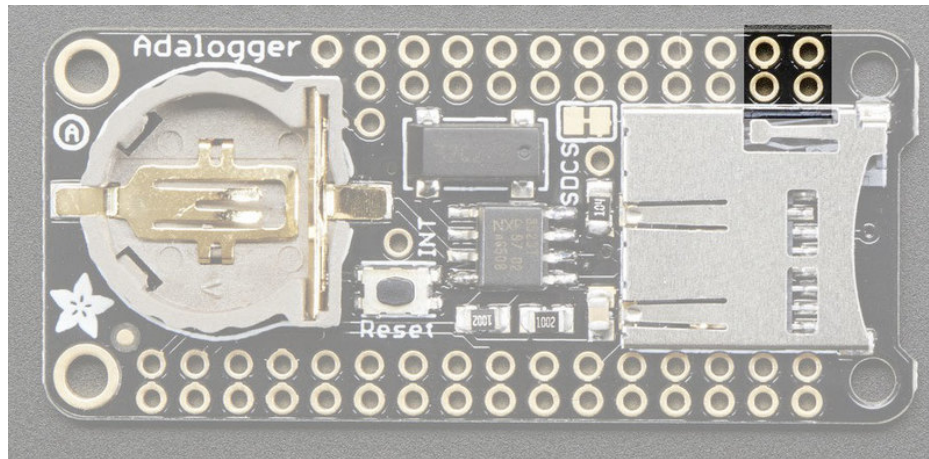
## Power Pins



On the bottom row, the **3.3V** (second from left) and **GND** (fourth 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

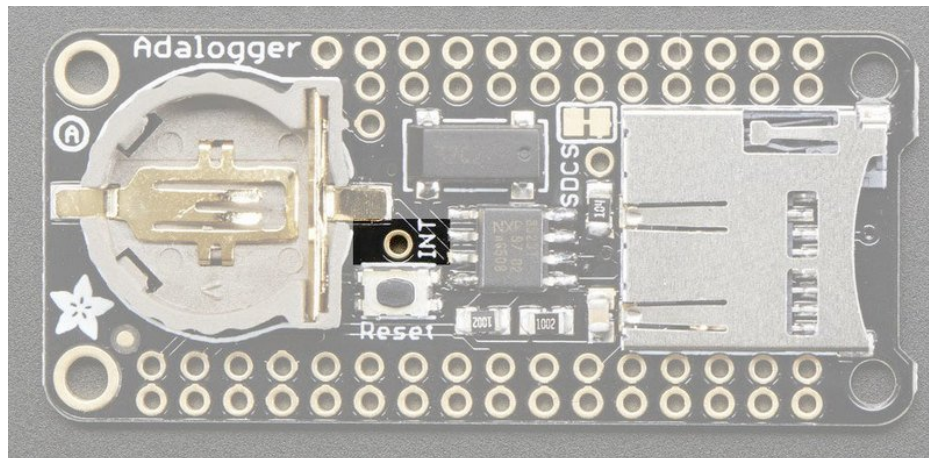




In the top right, **SDA** (rightmost) and **SCL** (to the left of SDA) are used to talk to the RTC chip.

- **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

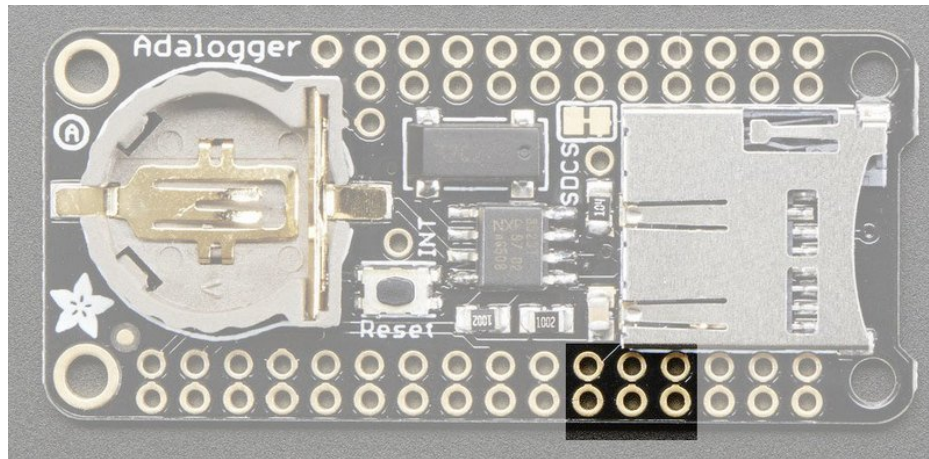


There's also a breakout for **INT** which is the output pin from the RTC. It can be used as an interrupt output or it could also be used to generate a square wave.

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

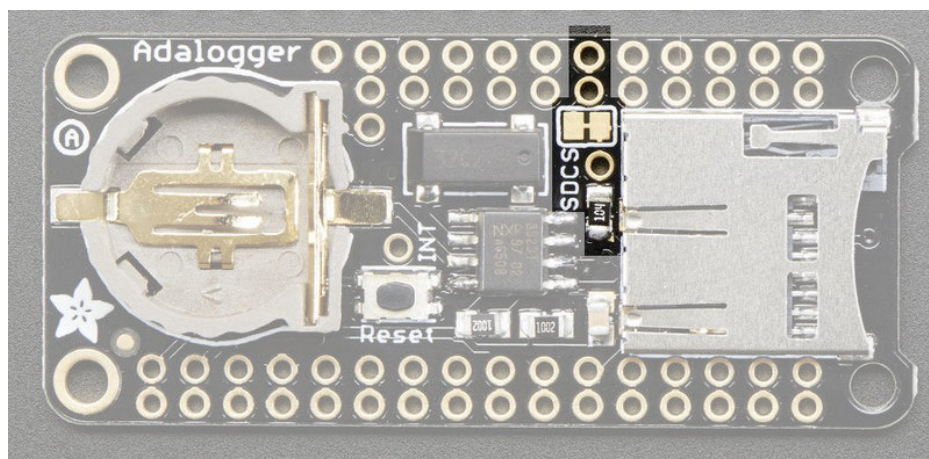
- **SPI Clock (SCK)** - output from feather to wing
- **SPI Microcontroller Out Sensor In (MOSI)** - output from feather to wing
- **SPI Microcontroller In Sensor Out (MISO)** - input from wing to feather

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 ESP32 it's GPIO **33**
- On WICED it's GPIO **PB5**
- On the nRF52832 it's GPIO **11**
- On Atmel M0, M4, 328p or 32u4 it's on GPIO **10**
- On Teensy 3.x it's on GPIO **10**

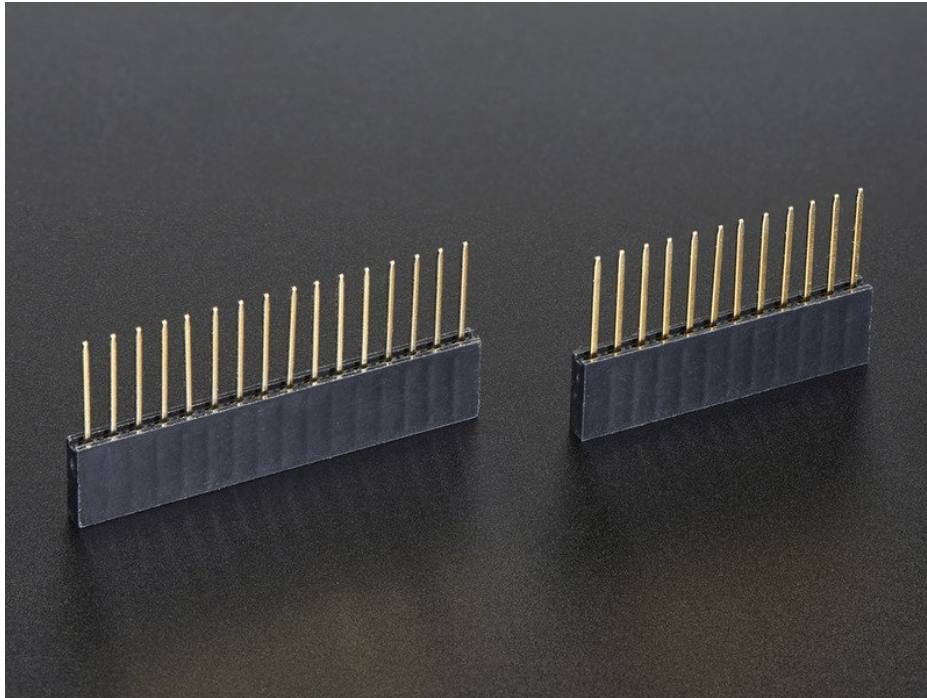
You can cut the trace to the default pin and change this to any pin by soldering a wire to any available pad.



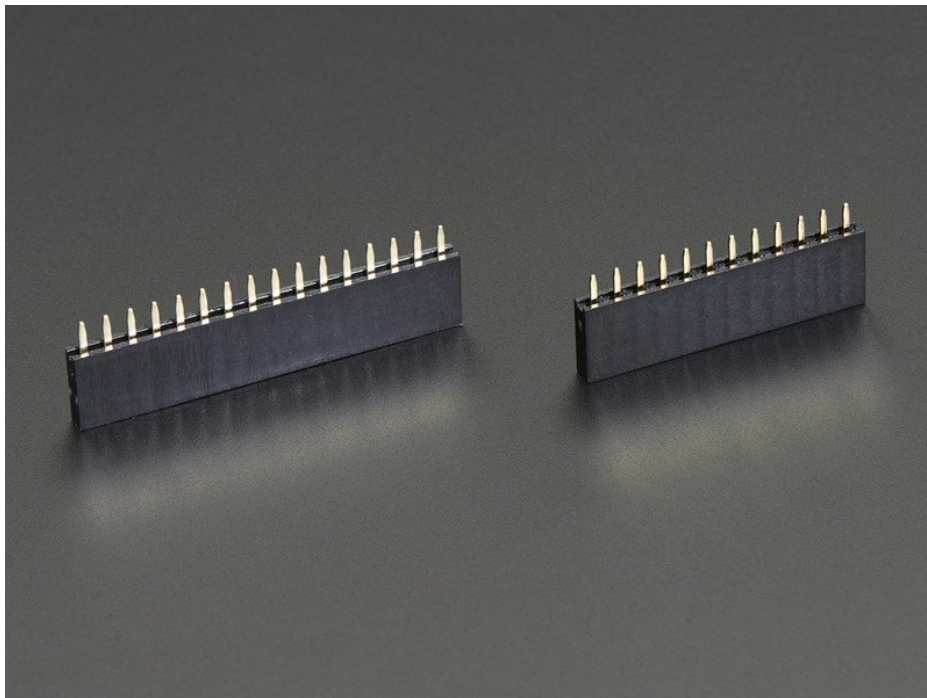


## Assembly

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

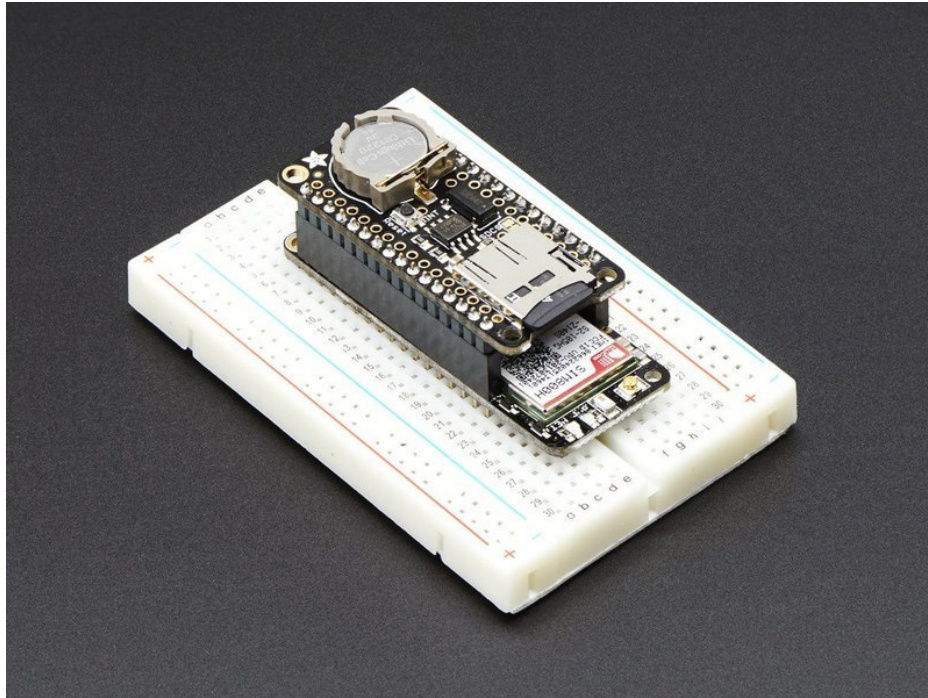


Or plain female socket headers:



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 there's 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 doesn't know its 'Tuesday' or 'March 8th' all it can tell is 'It's 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 doesn't 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](https://adafru.it/aX0) (<https://adafru.it/aX0>). There's a lithium battery in there which is why it's so big.*

The RTC we'll be using is the [PCF8523](https://adafru.it/reb) (<https://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:





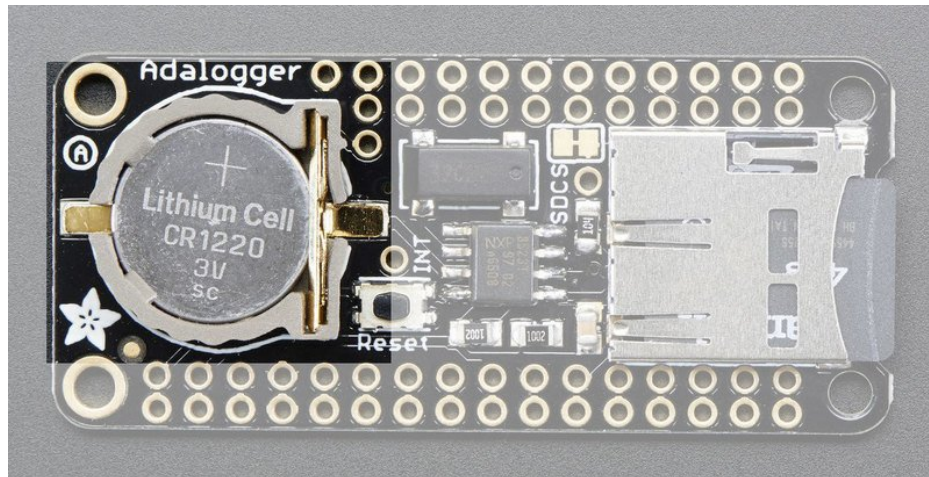
CR1220 12mm Diameter - 3V Lithium Coin Cell Battery

OUT OF STOCK

Out Of 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.



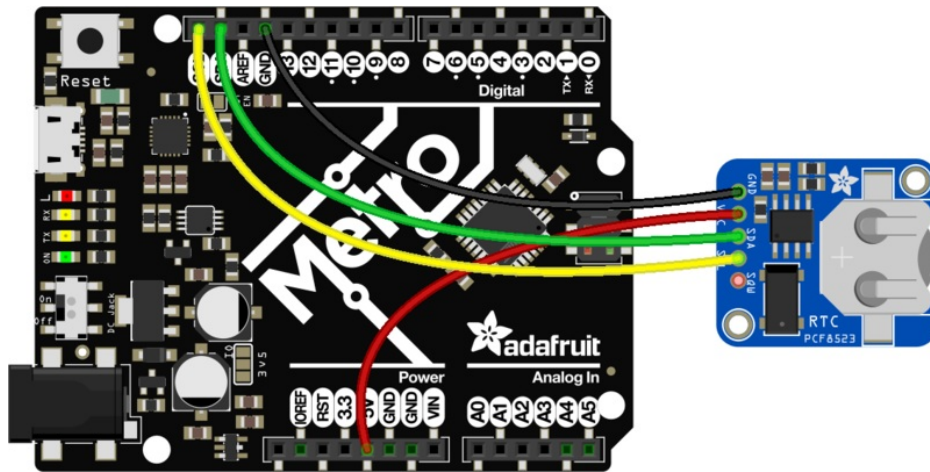
## RTC with Arduino

### Wiring

Wiring it up is easy, connect

- **GND** to **GND** on your board
- **VCC** to the logic level power of your board (on classic Arduinos & Metros use 5V, on 3.3V devices use 3.3V)
- **SDA** to the **SDA** i2c data pin
- **SCL** to the **SCL** i2c clock pin

There are internal 10K pull-ups on the PCF8523 on SDA and SCL to the VCC voltage



fritzing

<https://adafru.it/A1F>

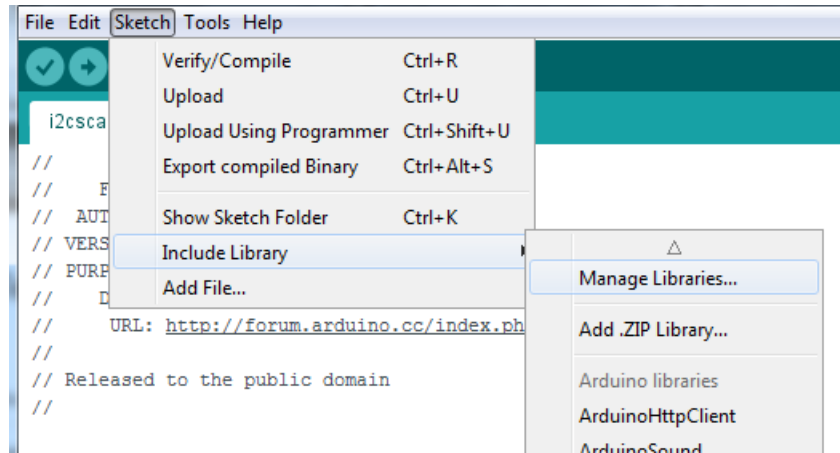
<https://adafru.it/A1F>

### Talking to the RTC

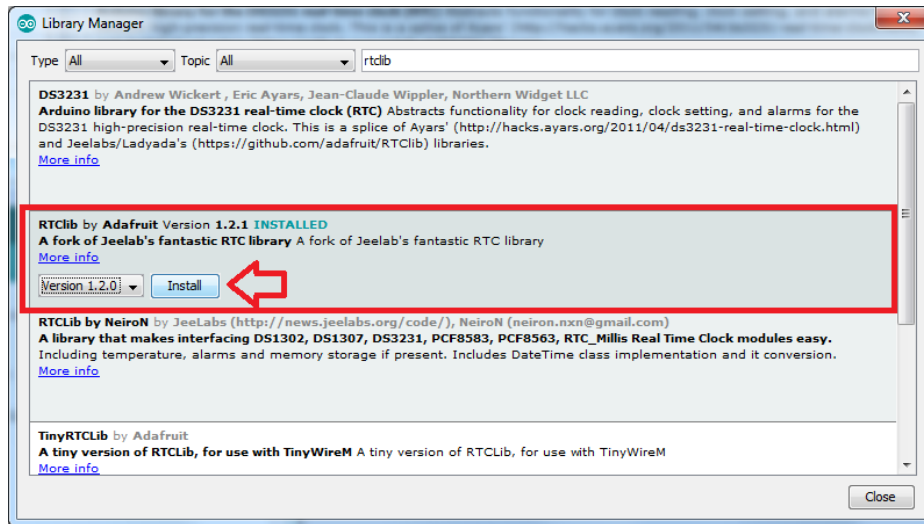
The RTC is an i2c device, which means it uses 2 wires to to communicate. These two wires are used to set the time and retrieve it.

For the RTC library, we'll be using a fork of JeeLab's excellent RTC library, [which is available on GitHub \(https://adafru.it/c7r\)](https://adafru.it/c7r). You can do that by visiting the github repo and manually downloading or, easier go to the **Arduino Library Manager**





Type in **RTCLib** - and find the one that is by **Adafruit** and click **Install**



There are a few different 'forks' of RTCLib, make sure you are using the ADAFRUIT one!

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

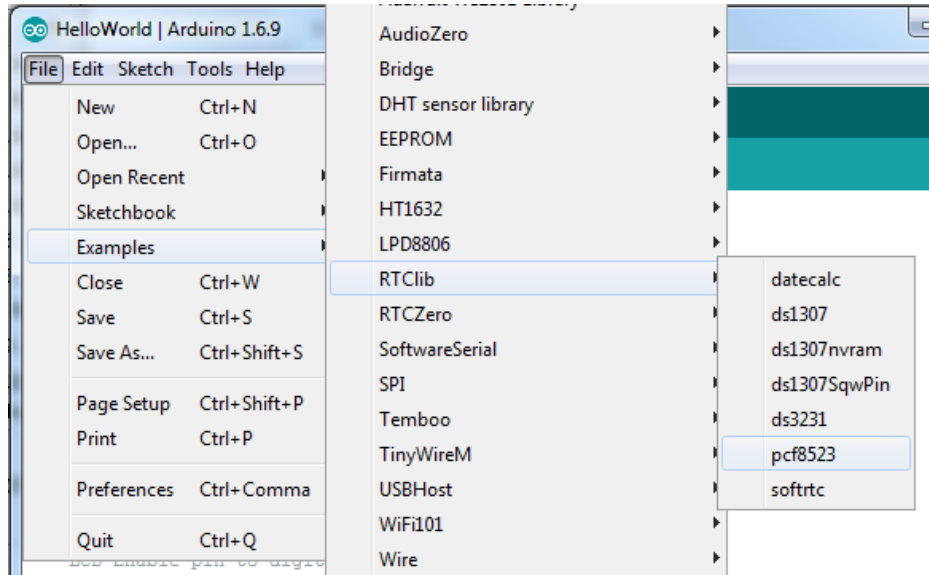
Once done, restart the IDE

## First RTC test

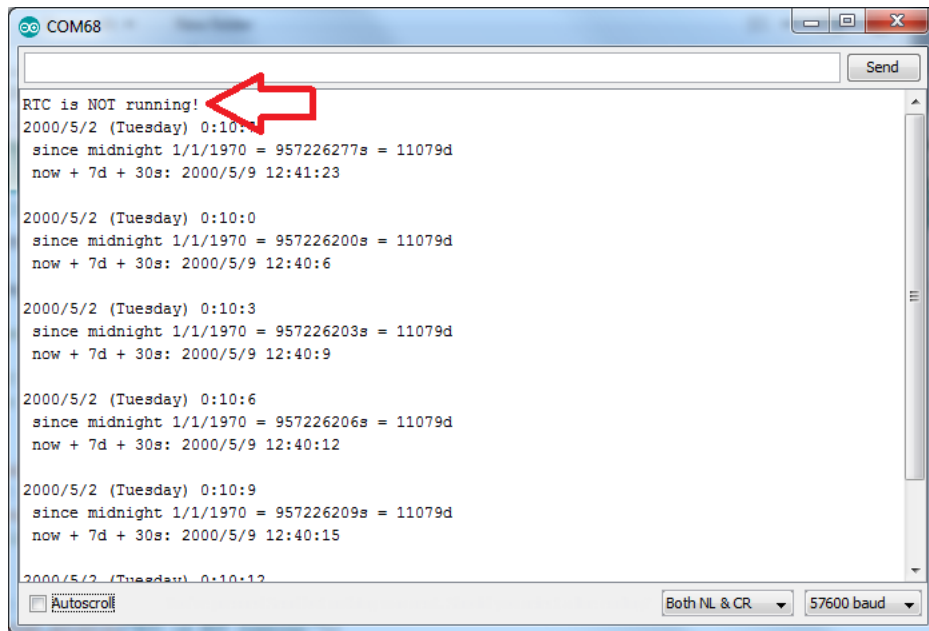
The first thing we'll demonstrate is a test sketch that will read the time from the RTC once a second. We'll also show what happens if you remove the battery and replace it since that causes the RTC to halt. So to start, remove the battery from the holder while the Feather is not powered or plugged into 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 board *with the PCF8523 breakout board or FeatherWing connected*



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



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

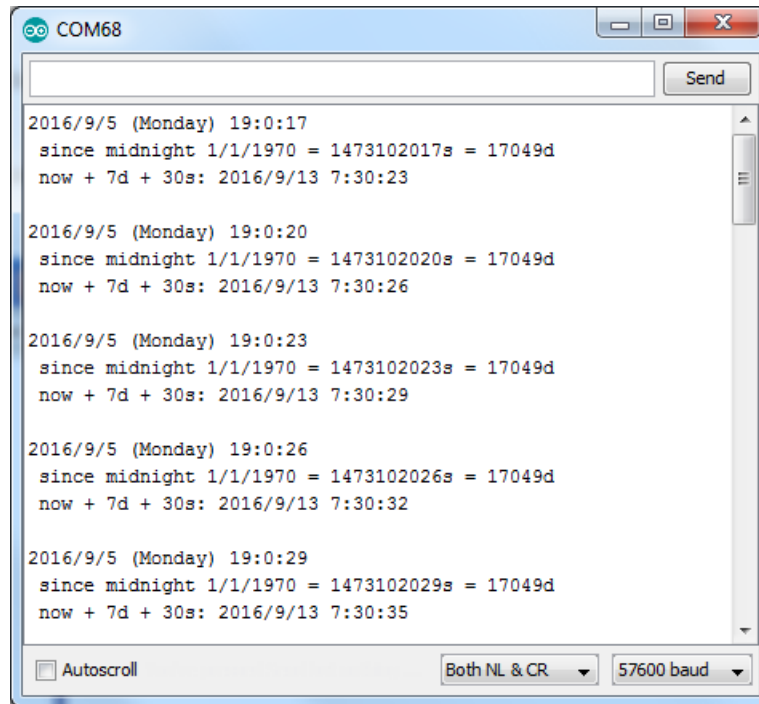
With the same sketch loaded, uncomment the line that starts with **RTC.adjust** like so:



```
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



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 occurrence - 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 it's 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");

```

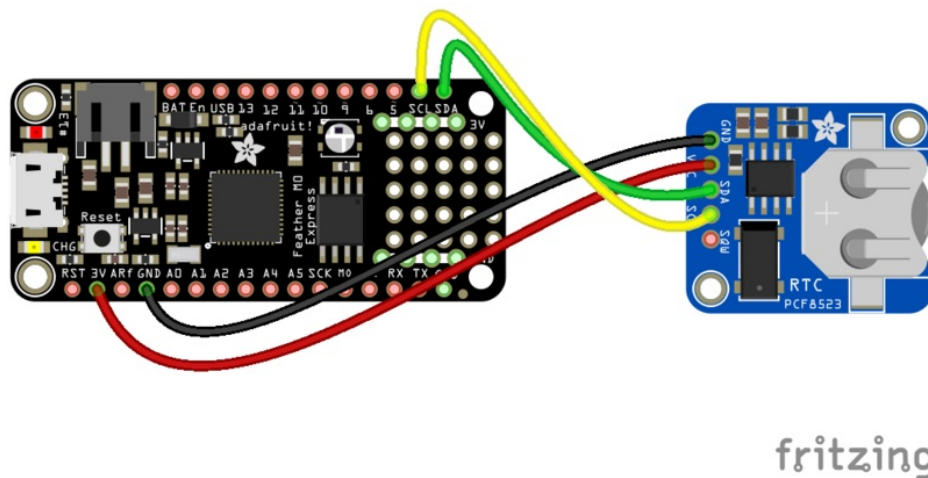
Since there are  $60 \times 60 \times 24 = 86400$  seconds in a day, we can easily count days since then as well. This might be useful when you want to keep track of how much time has passed since the last query, making some math a lot easier (like checking if it's been 5 minutes later, just see if **unixtime()** has increased by 300, you don't have to worry about hour changes)

## Wiring

Wiring it up is easy, connect

- **GND** to **GND** on your board
- **VCC** to the logic level power of your board - every CircuitPython board uses 3.3V
- **SDA** to the **SDA** i2c data pin
- **SCL** to the **SCL** i2c clock pin

There are internal 10K pull-ups on the PCF8523 on SDA and SCL to the VCC voltage



## Adafruit CircuitPython Library Install

To use the RTC sensor with your [Adafruit CircuitPython](https://adafru.it/AIP) (https://adafru.it/AIP) board you'll need to install the [Adafruit\\_CircuitPython\\_PCF8523](https://adafru.it/Bvh) (https://adafru.it/Bvh) module on your board.

First make sure you are running the [latest version of Adafruit CircuitPython](https://adafru.it/Amd) (https://adafru.it/Amd) for your board.

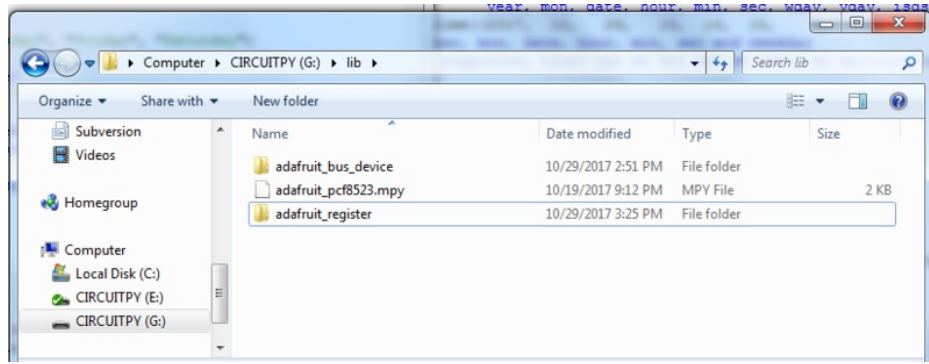
Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle](https://adafru.it/zdx) (https://adafru.it/zdx). Our introduction guide has [a great page on how to install the library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

Remember for non-express boards like the, you'll need to manually install the necessary libraries from the bundle:

- `adafruit_bus_device` folder
- `adafruit_register` folder
- `adafruit_pcf8523.mpy`

Before continuing make sure your board's `lib` folder or root filesystem has the `adafruit_pcf8523.mpy` module, the `adafruit_register` folder, and the `adafruit_bus_device` folder copied over.





## Usage

To demonstrate the usage of the PCF8523 module you can connect to your board's serial REPL to see the output while saving our example sketch to **main.py**

Next [connect to the board's serial REPL \(https://adafru.it/Awz\)](https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

Then save this script to **main.py** (back up or remove whatever was there before)

```
import busio
import adafruit_pcf8523
import time
import board

myI2C = busio.I2C(board.SCL, board.SDA)
rtc = adafruit_pcf8523.PCF8523(myI2C)

days = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")

if False:  # change to True if you want to write the time!
    #          year, mon, date, hour, min, sec, wday, yday, isdst
    t = time.struct_time((2017, 10, 29, 15, 14, 15, 0, -1, -1))
    # you must set year, mon, date, hour, min, sec and weekday
    # yearday is not supported, isdst can be set but we don't do anything with it at this time

    print("Setting time to:", t)  # uncomment for debugging
    rtc.datetime = t
    print()

while True:
    t = rtc.datetime
    #print(t)  # uncomment for debugging

    print("The date is %s %d/%d/%d" % (days[t.tm_wday], t.tm_mday, t.tm_mon, t.tm_year))
    print("The time is %d:%02d:%02d" % (t.tm_hour, t.tm_min, t.tm_sec))

    time.sleep(1) # wait a second
```

## Setting the time

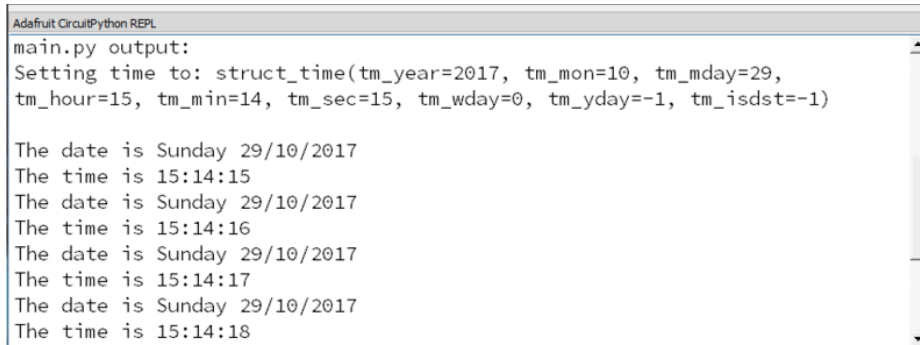
The first time you run the program, you'll need to **set the time**

find these lines:

```
if False:    # change to True if you want to write the time!
    #                year, mon, date, hour, min, sec, wday, yday, isdst
    t = time.struct_time((2017, 10, 29, 15, 14, 15, 0, -1, -1))
    # you must set year, mon, date, hour, min, sec and weekday
    # yearday is not supported, isdst can be set but we don't do anything with it at this time
```

Change the **False** to **True** in the first line, and update the `time.struct_time` to have the current time starting from `year` to `weekday`. The last two entries can stay at -1

Re-run the sketch by saving and you'll see this out of the REPL:



```
Adafruit CircuitPython REPL
main.py output:
Setting time to: struct_time(tm_year=2017, tm_mon=10, tm_mday=29,
tm_hour=15, tm_min=14, tm_sec=15, tm_wday=0, tm_yday=-1, tm_isdst=-1)

The date is Sunday 29/10/2017
The time is 15:14:15
The date is Sunday 29/10/2017
The time is 15:14:16
The date is Sunday 29/10/2017
The time is 15:14:17
The date is Sunday 29/10/2017
The time is 15:14:18
```

Note the part where the program says it is **Setting time to**:

Now you can go back and change the if **True** to if **False** and save, so you don't re-set the RTC again.

The script will now output the time and date

```
main.py output:
The date is Sunday 29/10/2017
The time is 15:14:56
The date is Sunday 29/10/2017
The time is 15:14:57
The date is Sunday 29/10/2017
The time is 15:14:58
```

## 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 but [we carry one in the shop that is guaranteed to work \(https://adafru.it/alH\)](https://adafru.it/alH). 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

OUT OF STOCK

Out Of 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 \(http://adafru.it/939\)](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

\$5.95  
IN STOCK

Add To Cart

## 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/](https://www.sdcard.org/downloads/formatter_4/) (<https://adafru.it/cfL>)

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

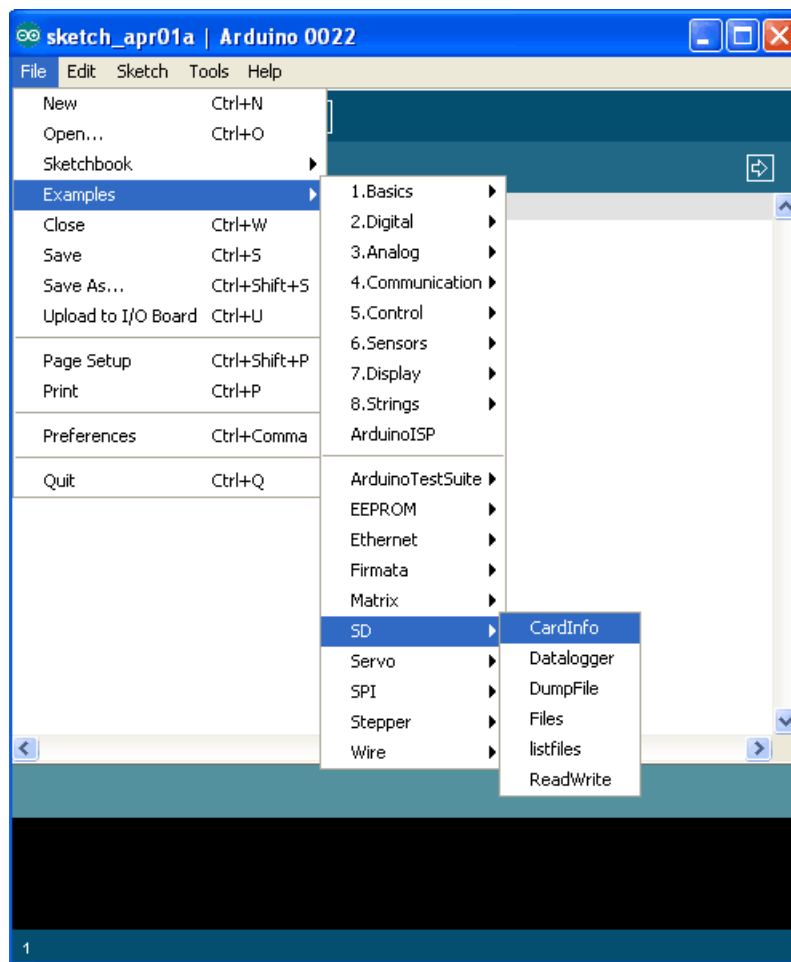
<https://adafru.it/cfL>

<https://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, M4, 328p or 32u4 it's on GPIO 10
- On Teensy 3.x it's on GPIO 10
- On STM32F2/WICED, it's on PB5
- On ESP32, it's on GPIO 33
- On nRF52832, it's on GPIO 11
- On nRF52840, it's on GPIO 10

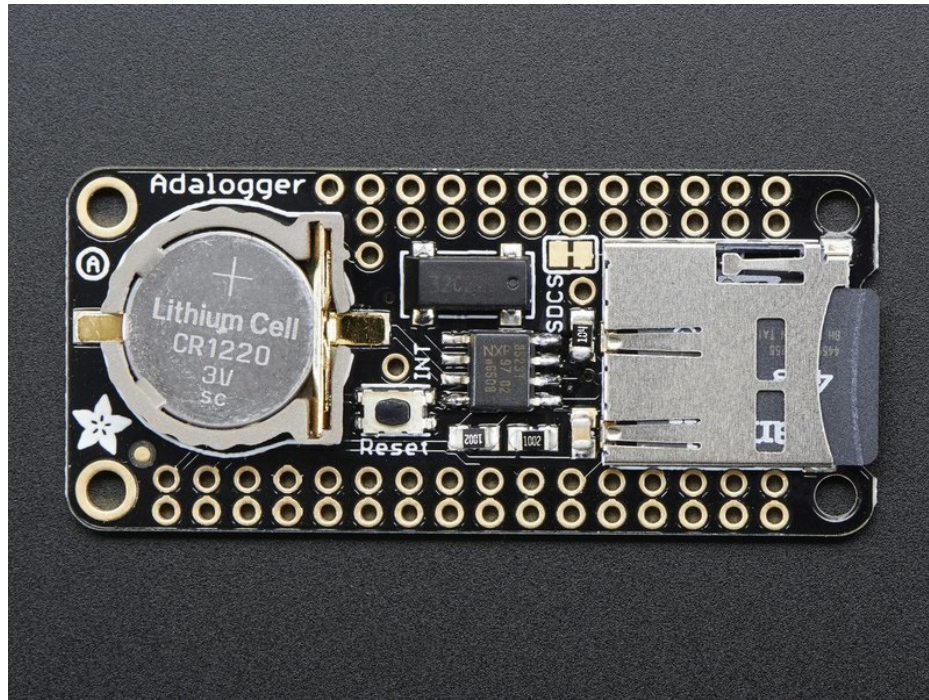


The screenshot shows the Arduino IDE interface with the 'CardInfo' sketch loaded. The title bar indicates 'CardInfo | Arduino 0022'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for running, stopping, saving, and other functions. The main text area displays the code for the 'CardInfo' sketch. A red rectangular box highlights the following code lines:

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

The code continues with a 'void setup()' function that initializes the serial port at 9600 baud, prints a message, and configures pin 10 as an output. It also includes a comment about using utility libraries for initialization. The status bar at the bottom shows the line number '31'.

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



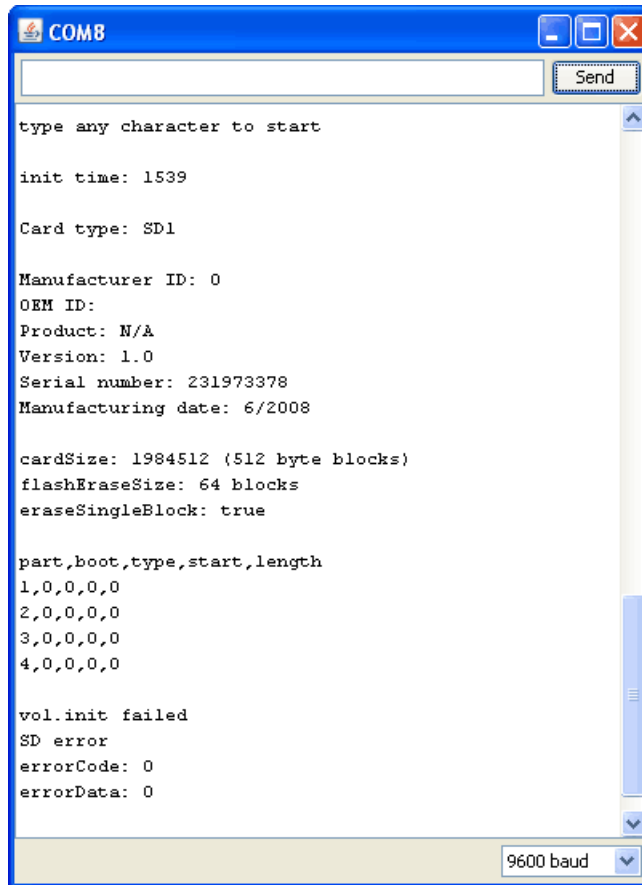
Open up the Serial Monitor and type in a character into the text box (& hit send) when prompted. You'll probably get something like the following:

```
COM53
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
Autoscroll No line ending 9600 baud
```

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.

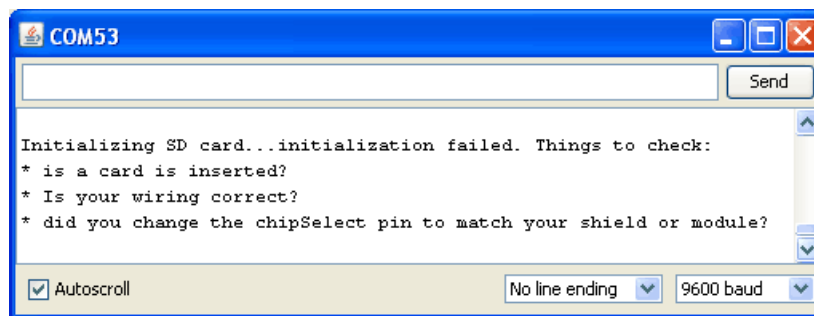
If you have a bad card, which seems to happen more with ripoff version of good brands, you might see:





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,



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!

## Next steps!

Once you know the SD card works, check out the [SD card library \(https://adafruit.it/ucu\)](https://adafruit.it/ucu) examples, [SD library](#)

## Example logging sketch

If you want to try saving data to the SD card in the simplest sketch, try this example. You can adjust the **delay()** to set how often analog data is read from pin **A0** and saved to the SD card. The red LED will blink if there's an error, and the green LED will blink when data is written to the SD card.



You will need to change the sketch's SD\_CS pin to match the SD card's Chip Select pin on your Feather!

```
1  #include <SPI.h>
2  #include <SD.h>
3
4  // Set the pins used
5  #define cardSelect 4
6
7  File logfile;
8
9  // blink out an error code
10 void error(uint8_t errno) {
11     while(1) {
12         uint8_t i;
13         for (i=0; i<errno; i++) {
14             digitalWrite(13, HIGH);
15             delay(100);
16             digitalWrite(13, LOW);
17             delay(100);
18         }
19         for (i=errno; i<10; i++) {
20             delay(200);
21         }
22     }
23 }
24
25 // This line is not needed if you have Adafruit SAMD board package 1.6.2+
26 // #define Serial SerialUSB
27
28 void setup() {
29     // connect at 115200 so we can read the GPS fast enough and echo without dropping chars
30     // also spit it out
31     Serial.begin(115200);
32     Serial.println("\r\nAnalog logger test");
33     pinMode(13, OUTPUT);
34
35
36     // see if the card is present and can be initialized:
37     if (!SD.begin(cardSelect)) {
38         Serial.println("Card init. failed!");
39         error(2);
40     }
41     char filename[15];
42     strcpy(filename, "/ANALOG00.TXT");
43     for (uint8_t i = 0; i < 100; i++) {
44         filename[7] = '0' + i/10;
```

```

45     filename[8] = '0' + i%10;
46     // create if does not exist, do not open existing, write, sync after write
47     if (! SD.exists(filename)) {
48         break;
49     }
50 }
51
52 logfile = SD.open(filename, FILE_WRITE);
53 if( ! logfile ) {
54     Serial.print("Couldnt create ");
55     Serial.println(filename);
56     error(3);
57 }
58 Serial.print("Writing to ");
59 Serial.println(filename);
60
61 pinMode(13, OUTPUT);
62 pinMode(8, OUTPUT);
63 Serial.println("Ready!");
64 }
65
66 uint8_t i=0;
67 void loop() {
68     digitalWrite(8, HIGH);
69     logfile.print("A0 = "); logfile.println(analogRead(0));
70     Serial.print("A0 = "); Serial.println(analogRead(0));
71     digitalWrite(8, LOW);
72
73     delay(100);
74 }

```

adalogger.ino hosted with ♥ by [GitHub](#)

[view raw](#)

If you really want to make sure you save every data point, put a

```
logfile.flush();
```

right after the `logfile.print`'s however this will cause the adalogger to draw a lot more power, maybe about 3x as much on average (30mA avg rather than about 10mA)



## Adafruit CircuitPython Module Install

To use a microSD card with your Adafruit CircuitPython board you'll need to install the [Adafruit\\_CircuitPython\\_SD](https://adafru.it/zwC) (<https://adafru.it/zwC>) module on your board.

First make sure you are running the [latest version of Adafruit CircuitPython](https://adafru.it/Amd) (<https://adafru.it/Amd>) for your board.

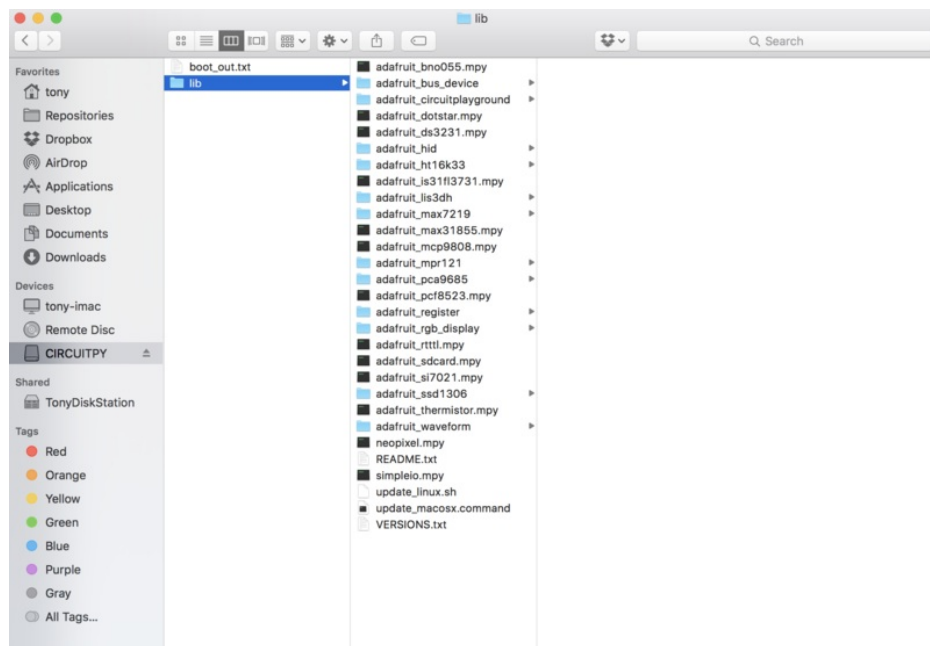
Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle](https://adafru.it/zdx) (<https://adafru.it/zdx>). Our introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<https://adafru.it/ABU>) for both express and non-express boards. **Be sure to use the latest CircuitPython Bundle** as it includes an updated version of the SD card module with a few necessary fixes!

Remember for non-express boards like the, you'll need to manually install the necessary libraries from the bundle:

- `adafruit_sdcard.mpy`
- `adafruit_bus_device`

If your board doesn't support USB mass storage, like the ESP8266, then [use a tool like ampy to copy the file to the board](https://adafru.it/s1f) (<https://adafru.it/s1f>). You can use the latest version of ampy and its [new directory copy command](https://adafru.it/q2A) (<https://adafru.it/q2A>) to easily move module directories to the board.

Before continuing make sure your board's lib folder or root filesystem has the `adafruit_sdcard.mpy` and `adafruit_bus_device` modules copied over.



## Usage

The following section will show how to initialize the SD card and read & write data to it from the board's Python prompt / REPL.

Next [connect to the board's serial REPL \(https://adafru.it/Awz\)](https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

## Initialize & Mount SD Card Filesystem

Before you can use the microSD card you need to initialize its SPI connection and mount its filesystem. First import the necessary modules to initialize the SPI and CS line physical connections:

```
import board
import busio
import digitalio
```

Next create the SPI bus and a digital output for the microSD card's chip select line (be sure to select the right pin name or number for your wiring):

```
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
# Use board.SD_CS for Feather M0 Adalogger
cs = digitalio.DigitalInOut(board.SD_CS)
# Or use a GPIO pin like 15 for ESP8266 wiring:
#cs = digitalio.DigitalInOut(board.GPI015)
```

Now import modules to access the SD card and filesystem:

```
import adafruit_sdcard
import storage
```

At this point you're ready to create the microSD card object and the filesystem object:

```
sdcard = adafruit_sdcard.SDCard(spi, cs)
vfs = storage.VfsFat(sdcard)
```

Notice the **adafruit\_sdcard** module has a **SDCard** class which contains all the logic for talking to the microSD card at a low level. This class needs to be told the SPI bus and chip select digital IO pin in its initializer.

After a **SDCard** class is created it can be passed to the **storage** module's **VfsFat** class. This class has all the logic for translating CircuitPython filesystem calls into low level microSD card access. Both the **SDCard** and **VfsFat** class instances are required to mount the card as a new filesystem.

Finally you can mount the microSD card's filesystem into the CircuitPython filesystem. For example to make the path **/sd** on the CircuitPython filesystem read and write from the card run this command:

```
storage.mount(vfs, "/sd")
```

The first parameter to the `storage.mount` command is the **VfsFat** class instance that was created above, and the second parameter is the location within the CircuitPython filesystem that you'd like to 'place' the microSD card. Remember the mount location as you'll need it to read and write files on the card!

## Reading & Writing Data

Once the microSD card is mounted inside CircuitPython's filesystem you're ready to read and write data from it.

Reading and writing data is simple using Python's file operations like [open \(https://adafru.it/rel\)](https://adafru.it/rel), [close \(https://adafru.it/ryE\)](https://adafru.it/ryE), [read \(https://adafru.it/ryE\)](https://adafru.it/ryE), and [write \(https://adafru.it/ryE\)](https://adafru.it/ryE). The beauty of CircuitPython and MicroPython is that they try to be as similar to desktop Python as possible, including access to files.

For example to create a file and write a line of text to it you can run:

```
with open("/sd/test.txt", "w") as f:
    f.write("Hello world!\r\n")
```

Notice the `with` statement is used to create a context manager that opens and automatically closes the file. This is handy because with file access you Python you **must** close the file when you're done or else all the data you thought was written might be lost!

The **open** function is used to open the file by telling it the path to it, and the mode (`w` for writing). Notice the path is under `/sd`, `/sd/test.txt`. This means the file will be created on the microSD card that was mounted as that path.

Inside the context manager you can access the `f` variable to operate on the file while it's open. The **write** function is called to write a line of text to the file. Notice that unlike a print statement you need to end the string passed to write with explicit carriage returns and new lines.

You can also open a file and read a line from it with similar code:

```
with open("/sd/test.txt", "r") as f:
    print("Read line from file:")
    print(f.readline())
```

If you wanted to read and print all of the lines from a file you could call **readline** in a loop. Once **readline** reaches the end of the file it will return an empty string so you know when to stop:

```
with open("/sd/test.txt", "r") as f:
    print("Printing lines in file:")
    line = f.readline()
    while line != '':
        print(line)
        line = f.readline()
```

There's even a **readlines** function that will read all of the lines in the file and return them in an array of lines. Be careful though as this means the entire file must be loaded into memory, so if the file is very large you might run out of memory. If you know your file is very small you can use it though:

```
with open("/sd/test.txt", "r") as f:
    lines = f.readlines()
    print("Printing lines in file:")
    for line in lines:
        print(line)
```

Finally one other very common file scenario is opening a file to add new data at the end, or append data. This works exactly the same as in Python and the **open** function can be told you'd like to append instead of erase and write new data (what normally happens with the `w` option for **open**). For example to add a line to the file:

```
with open("/sd/test.txt", "a") as f:  
    f.write("This is another line!\r\n")
```

Notice the **a** option in the open function--this tells Python to add data at the end of the file instead of erasing it and starting over at the top. Try reading the file with the code above to see the new line that was added!

That's all there is to manipulating files on microSD cards with CircuitPython!

Here are a few more complete examples of using a SD card from the [Trinket M0 CircuitPython guides \(https://adafru.it/Bvi\)](https://adafru.it/Bvi). These are great as a reference for more SD card usage.

## List Files

---

Load this into **main.py**:



```

import os

import adafruit_sdcard
import board
import busio
import digitalio
import storage

# Use any pin that is not taken by SPI
SD_CS = board.D0

# Connect to the card and mount the filesystem.
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
cs = digitalio.DigitalInOut(SD_CS)
sdcard = adafruit_sdcard.SDCard(spi, cs)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")

# Use the filesystem as normal! Our files are under /sd

# This helper function will print the contents of the SD

def print_directory(path, tabs=0):
    for file in os.listdir(path):
        stats = os.stat(path + "/" + file)
        filesize = stats[6]
        isdir = stats[0] & 0x4000

        if filesize < 1000:
            sizestr = str(filesize) + " by"
        elif filesize < 1000000:
            sizestr = "%0.1f KB" % (filesize / 1000)
        else:
            sizestr = "%0.1f MB" % (filesize / 1000000)

        prettyprintname = ""
        for _ in range(tabs):
            prettyprintname += "  "
        prettyprintname += file
        if isdir:
            prettyprintname += "/"
        print('{0:<40} Size: {1:>10}'.format(prettyprintname, sizestr))

        # recursively print directory contents
        if isdir:
            print_directory(path + "/" + file, tabs + 1)

print("Files on filesystem:")
print("=====")
print_directory("/sd")

```

Once it's loaded up, open up the REPL (and restart it with ^D if necessary) to get a printout of all the files included. We recursively print out all files and also the filesize. This is a good demo to start with because you can at least tell if your files exist!

| Adafruit CircuitPython REPL |       |          |
|-----------------------------|-------|----------|
| Files on filesystem:        |       |          |
| =====                       |       |          |
| TeensyDemo.bin              | Size: | 8.4 MB   |
| SEARCH.HTM                  | Size: | 75.5 KB  |
| fw.bin                      | Size: | 18.0 KB  |
| System Volume Information/  | Size: | 0 by     |
| WPSettings.dat              | Size: | 12 by    |
| IndexerVolumeGuid           | Size: | 76 by    |
| test.txt~                   | Size: | 254 by   |
| test.txt                    | Size: | 12 by    |
| binaries/                   | Size: | 0 by     |
| 2772copy.bin                | Size: | 239.6 KB |
| 2772test.bin                | Size: | 29.6 KB  |
| bootload.bin                | Size: | 8.2 KB   |
| 2772blnk.bin                | Size: | 18.0 KB  |

## Logging Temperature

But you probably want to do a little more, lets log the temperature from the chip to a file.

Here's the new script

```
import time

import adafruit_sdcard
import board
import busio
import digitalio
import microcontroller
import storage

# Use any pin that is not taken by SPI
SD_CS = board.D0

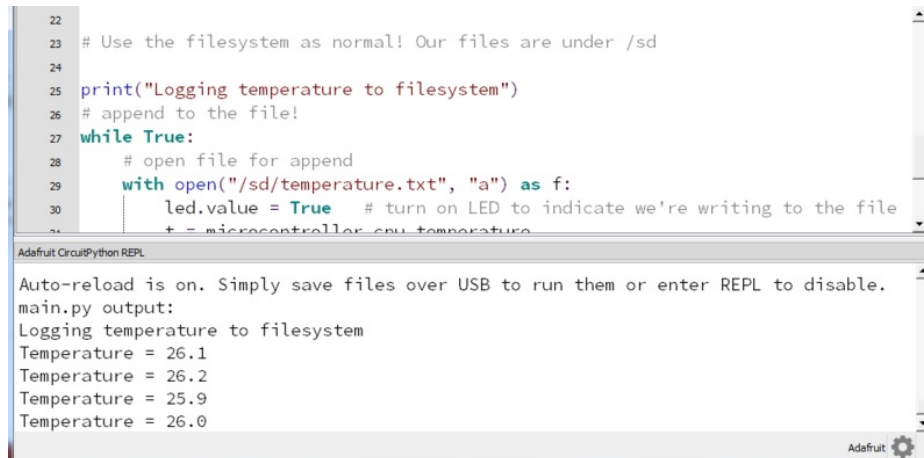
led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

# Connect to the card and mount the filesystem.
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
cs = digitalio.DigitalInOut(SD_CS)
sdcard = adafruit_sdcard.SDCard(spi, cs)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")

# Use the filesystem as normal! Our files are under /sd

print("Logging temperature to filesystem")
# append to the file!
while True:
    # open file for append
    with open("/sd/temperature.txt", "a") as f:
        led.value = True # turn on LED to indicate we're writing to the file
        t = microcontroller.cpu.temperature
        print("Temperature = %0.1f" % t)
        f.write("%0.1f\n" % t)
        led.value = False # turn off LED to indicate we're done
    # file is saved
    time.sleep(1)
```

When saved, the Trinket will start saving the temperature once per second to the SD card under the file `temperature.txt`



The screenshot shows the Adafruit CircuitPython REPL interface. The top pane displays Python code for logging temperature to an SD card. The bottom pane shows the output of the program, including a confirmation message and a series of temperature readings.

```
22
23 # Use the filesystem as normal! Our files are under /sd
24
25 print("Logging temperature to filesystem")
26 # append to the file!
27 while True:
28     # open file for append
29     with open("/sd/temperature.txt", "a") as f:
30         led.value = True # turn on LED to indicate we're writing to the file
31         t = microcontroller.cpu.temperature
32         print("Temperature = %0.1f" % t)
33         f.write("%0.1f\n" % t)
34         led.value = False # turn off LED to indicate we're done
35     # file is saved
36     time.sleep(1)
```

Adafruit CircuitPython REPL

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.  
main.py output:  
Logging temperature to filesystem  
Temperature = 26.1  
Temperature = 26.2  
Temperature = 25.9  
Temperature = 26.0

The key part of this demo is in these lines:

```
print("Logging temperature to filesystem")
# append to the file!
while True:
    # open file for append
    with open("/sd/temperature.txt", "a") as f:
        led.value = True # turn on LED to indicate we're writing to the file
        t = microcontroller.cpu.temperature
        print("Temperature = %0.1f" % t)
        f.write("%0.1f\n" % t)
        led.value = False # turn off LED to indicate we're done
    # file is saved
    time.sleep(1)
```

This is a slightly complex demo but it's for a good reason. We use **with** (a 'context') to open the file for appending, that way the file is only opened for the very short time its written to. This is safer because then if the SD card is removed or the board turned off, all the data will be safe(r).

We use the LED to let the person using this know that the temperature is being written, it turns on just before the write and then off right after.

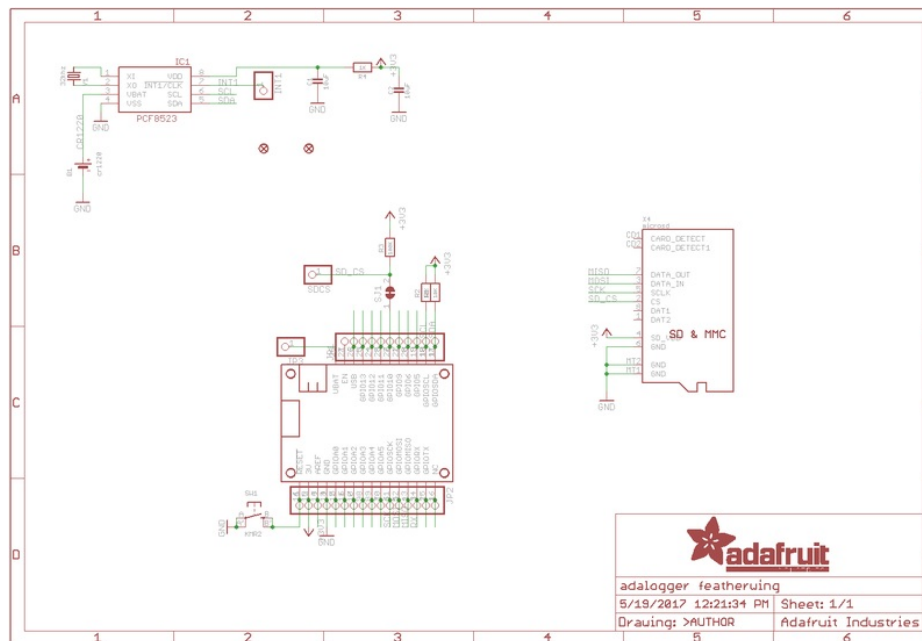
After the LED is turned off the **with** ends and the context closes, the file is safely stored.

## Downloads

## Datasheets and Files

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

## Schematic



## Fabrication Print

