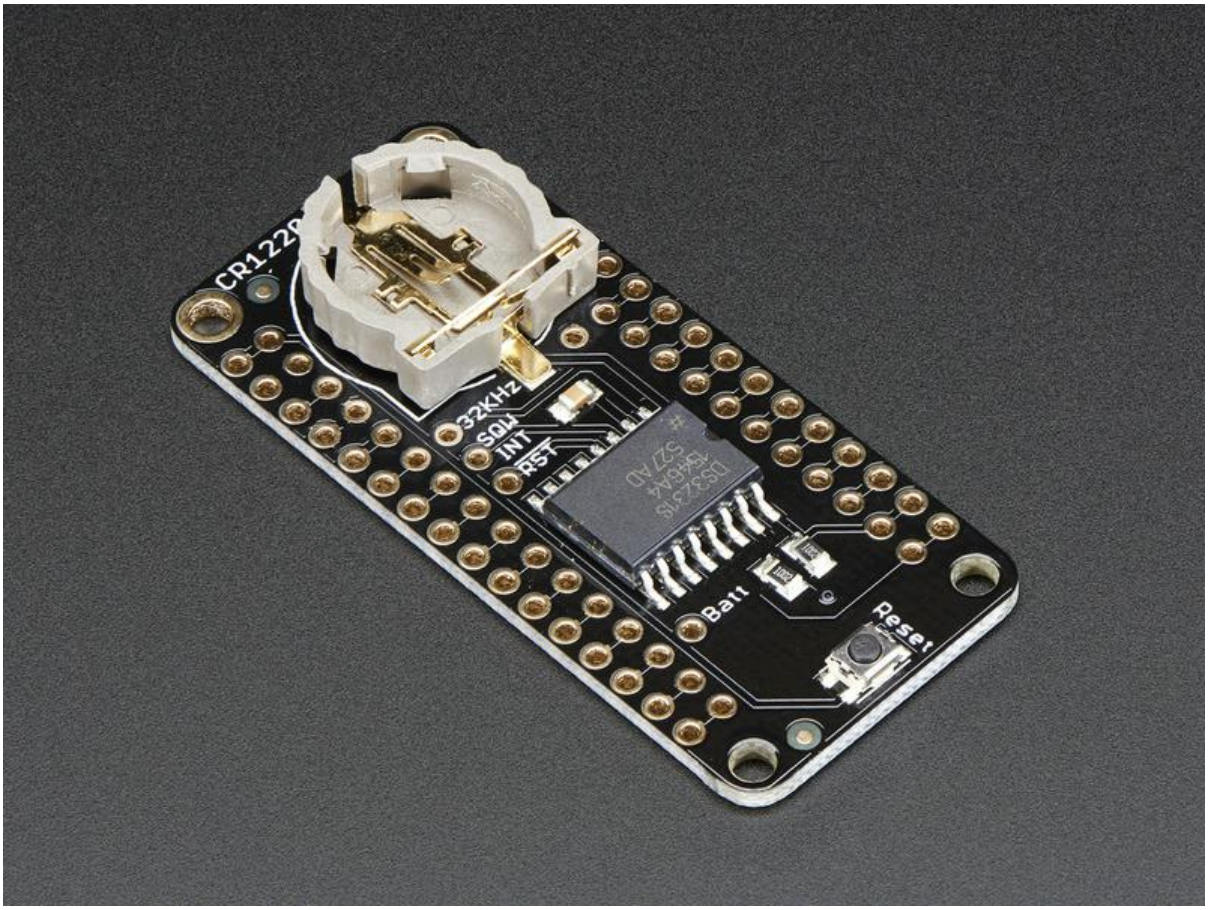
DS3231 Precision RTC FeatherWing

Created by lady ada

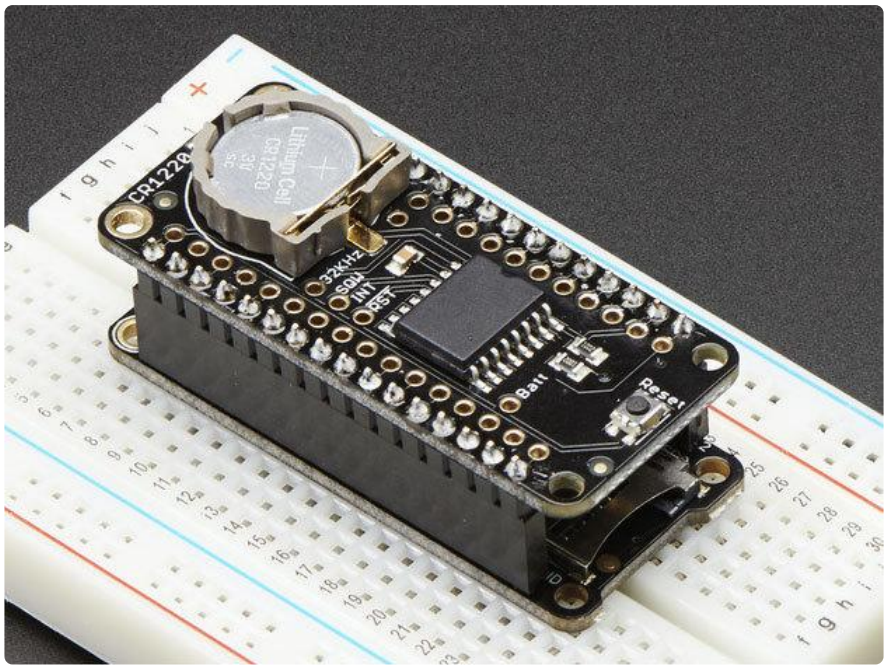


https://learn.adafruit.com/ds3231-precision-rtc-featherwing

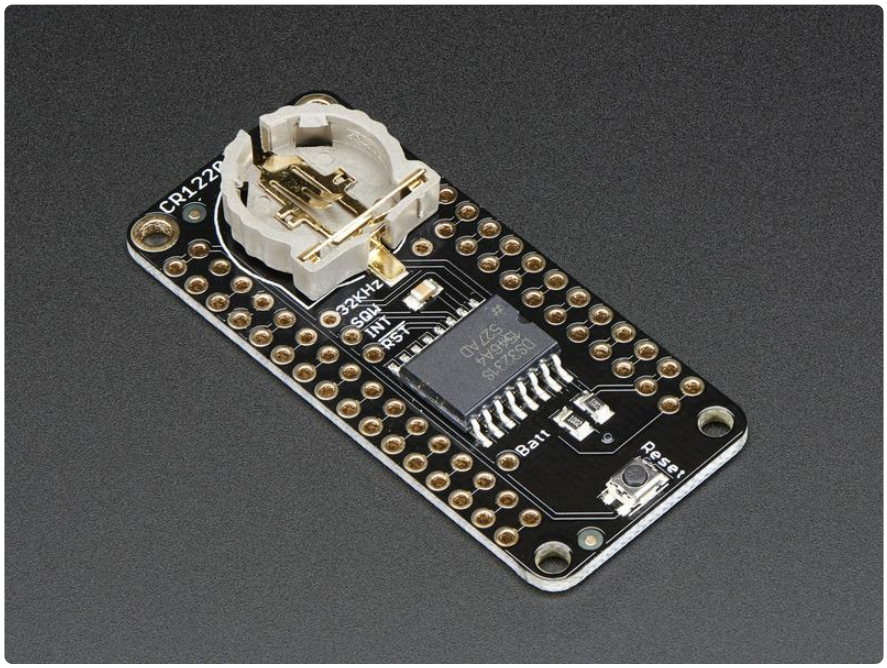
Last updated on 2021-11-15 06:39:52 PM EST

Table of Contents

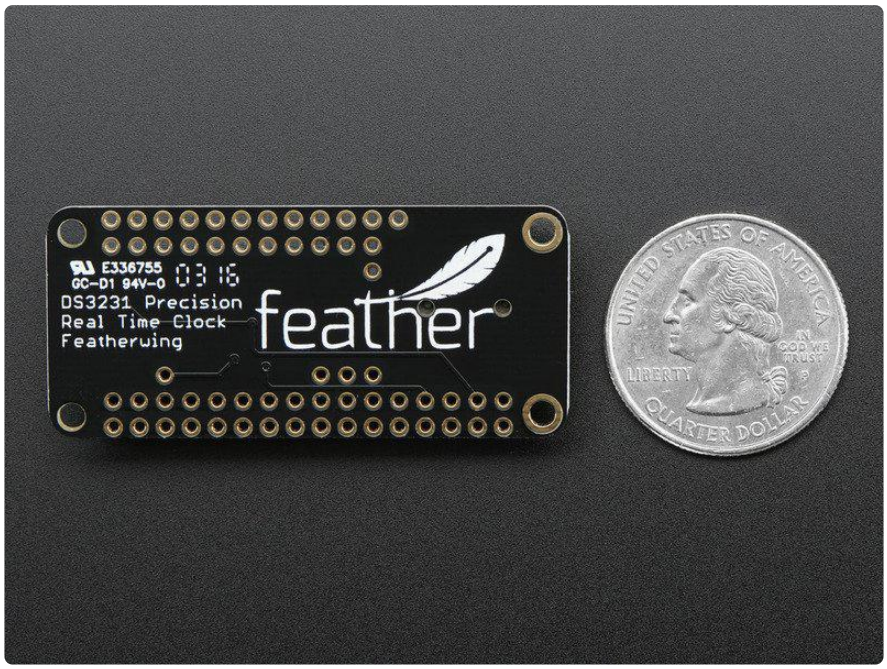
Overview



A Feather board without ambition is a Feather board without FeatherWings! This is the DS3231 Precision RTC FeatherWing: it adds an extremely accurate I2C-integrated Real Time Clock (RTC) with a Temperature Compensated Crystal Oscillator (TCXO) to any Feather main board. This RTC is the most precise you can get in a small, low power package. 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!

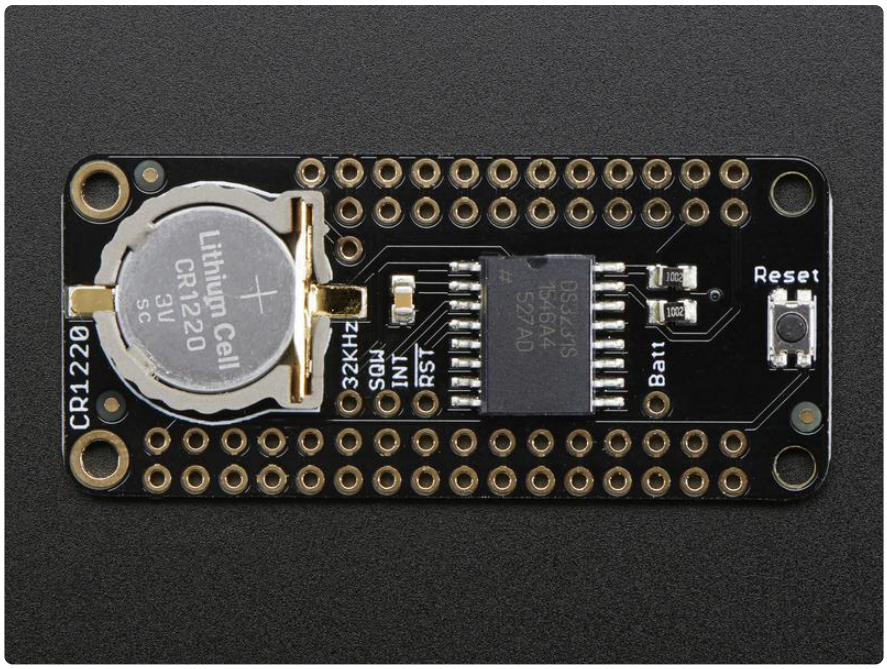


Most RTCs use an external 32kHz timing crystal that is used to keep time with low current draw. And that's all well and good, but those crystals have slight drift, particularly when the temperature changes (the temperature changes the oscillation frequency very very very slightly but it does add up!) This RTC is in a beefy package because the crystal is inside the chip! And right next to the integrated crystal is a temperature sensor. That sensor compensates for the frequency changes by adding or removing clock ticks so that the timekeeping stays on schedule.

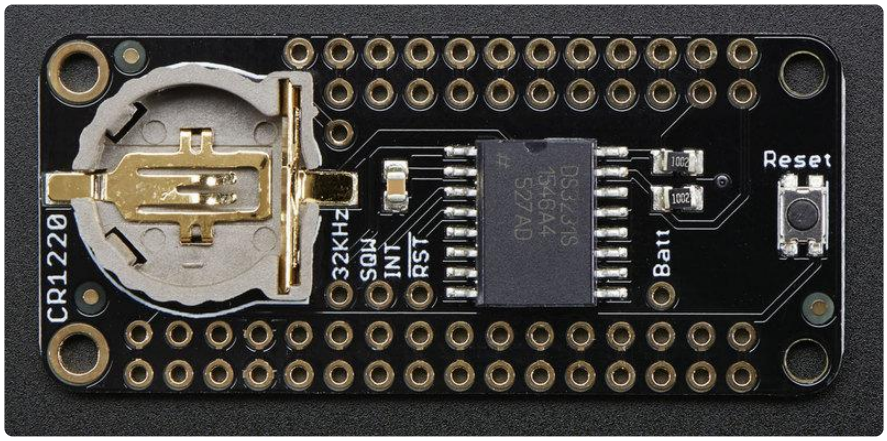


With a CR1220 12mm coin cell (http://adafru.it/380) plugged into the top of the FeatherWing, you can get years of precision timekeeping, even when main power is lost. Great for datalogging and clocks, or anything where you need to really know the time.

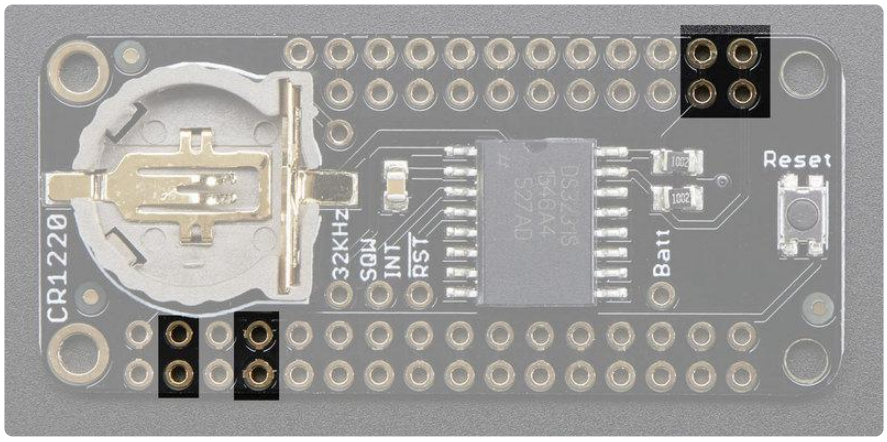
A CR1220 coin cell is required to use the battery-backup capabilities! We don't include one by default, to make shipping easier for those abroad, but we do stock them so pick one up or use any CR1220 you have handy. (http://adafru.it/380)



Pinouts



Power and I2C pins



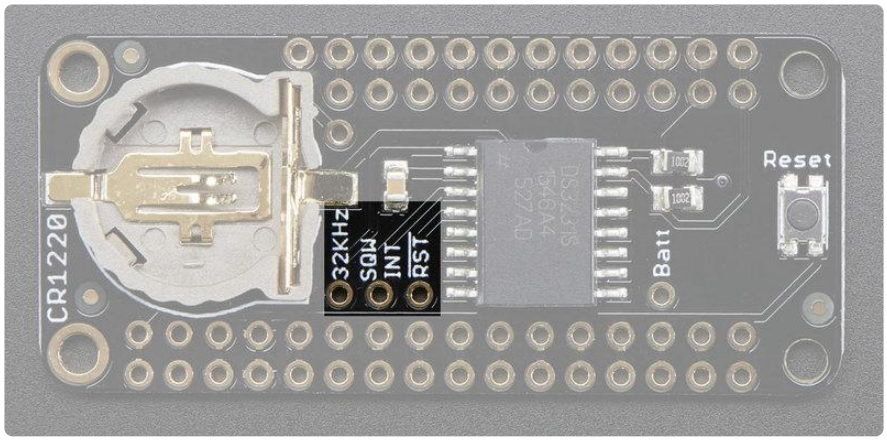
Even though every pin from the Feather is 'doubled up' with an inner header, only 4 of those pins are actually used!

On the bottom row, the 3.3V and GND pin are used to power the RTC - to take a load off the coin cell battery.

In the top right, SDA and SCL (i2c bus) are used to talk to the 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

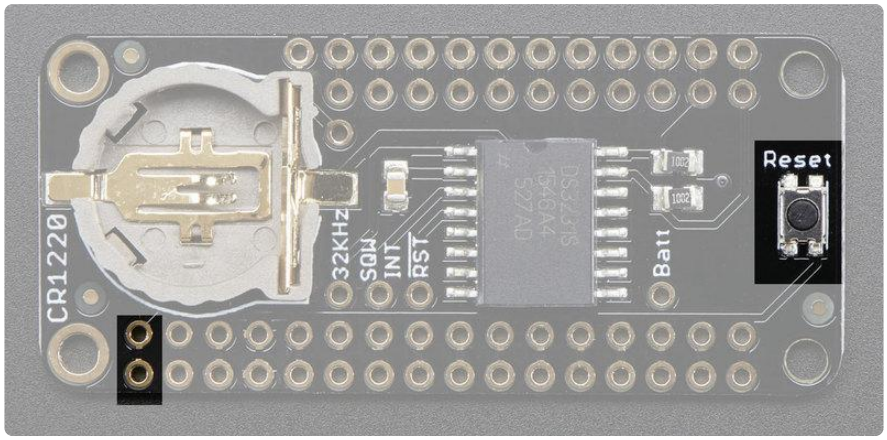
DS3231 Breakouts



The DS3231 has a few other pins you may want to use, they are broken out to a 3-pin header in the center.

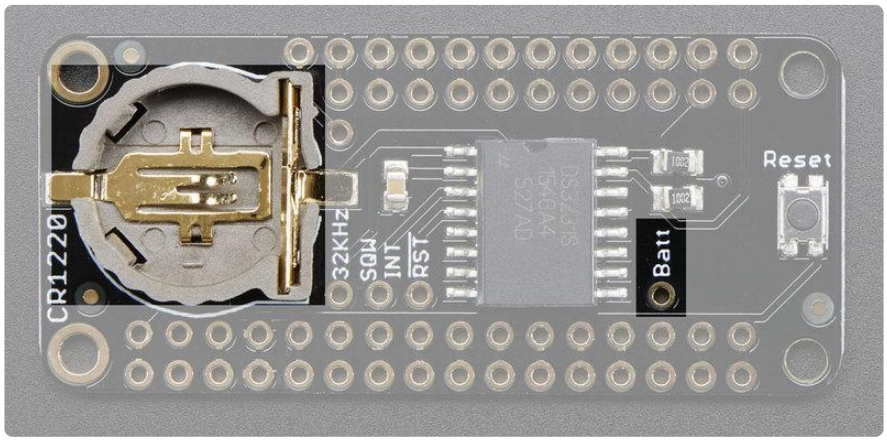
- 32K - 32KHz oscillator output. Open drain, you need to attach a pullup to read this signal from a microcontroller pin (or use a microcontroller that can turn on it's internal pullup)
- SQW - optional square wave or interrupt output. Open drain, you need to attach a pullup to read this signal from a microcontroller pin (or use a microcontroller that can turn on it's internal pullup)
- RST - This one is a little different than most RST pins, rather than being just an input, it is designed to be used to reset an external device or indicate when main power is lost. Open drain, but has an internal 50K pullup. The pullup keeps this pin voltage high as long as Vin is present. When Vin drops and the chip switches to battery backup, the pin goes low.

Reset



The Feather RESET pin is connected to a button on the right of the wing, handy if you need to reset or restart your Feather! Note that this RESET is not connected to the RS T pin on the DS3231

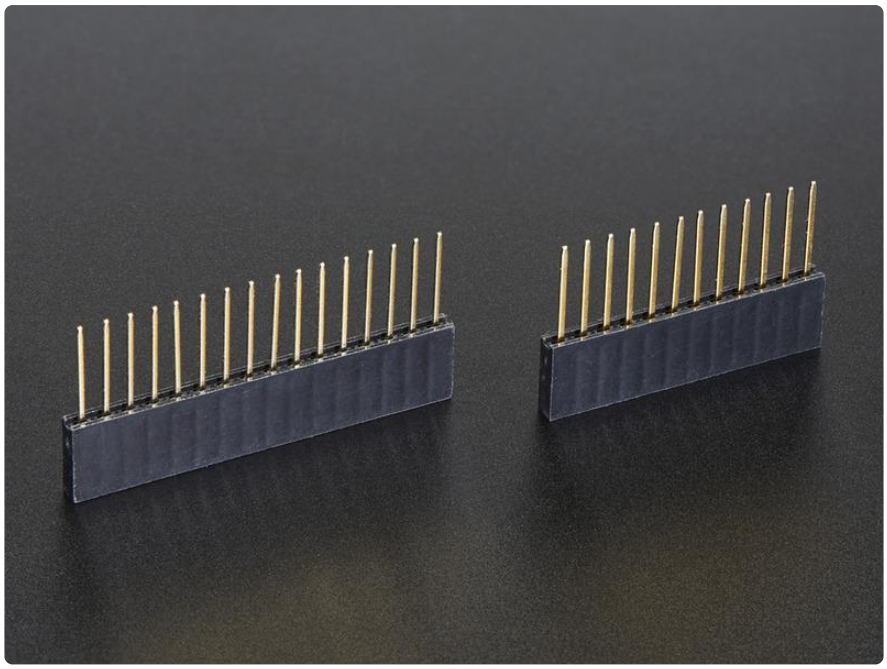
Battery



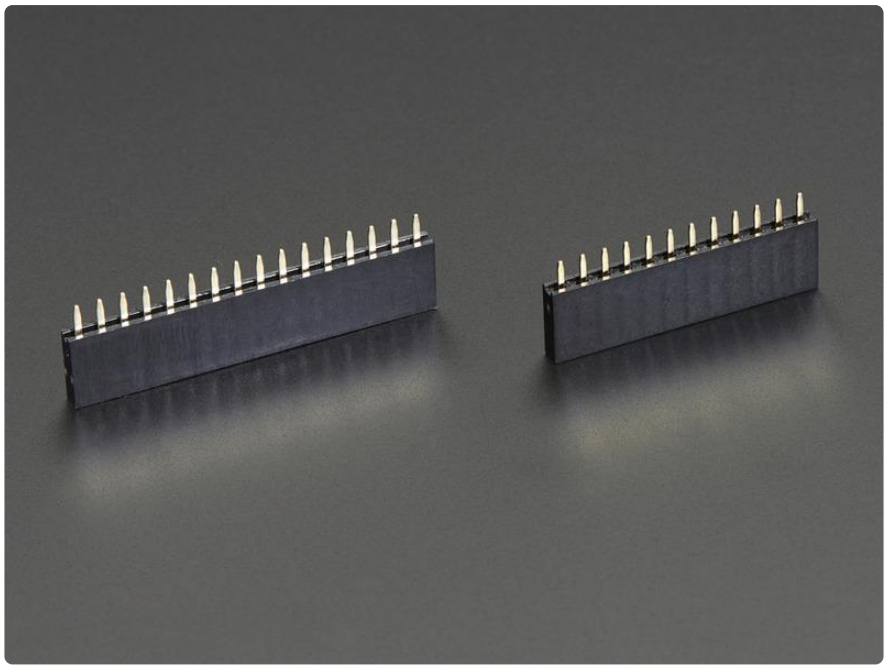
A CR1220 battery holder is on-board for long term RTC timekeeping. A new CR1220 should be able to power the DS3231 for at least 5 years. If you want to use that battery for powering something else (maybe a GPS module or a very low power BLE beacon?) you can connect to the BATT breakout on the very right.

Assembly

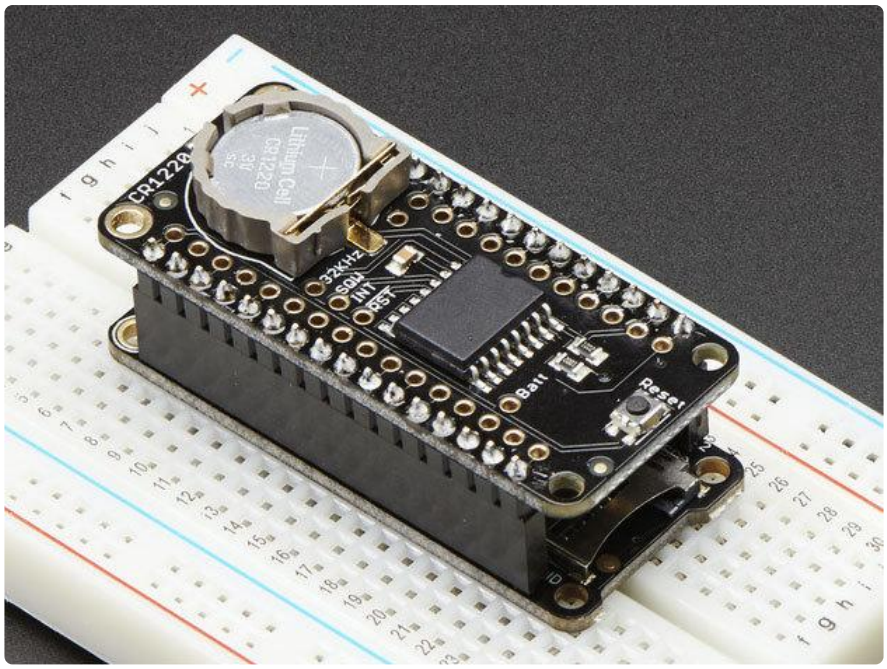
When putting together your Featherwings, think about how you want it to connect, you can use stacking headers:



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:

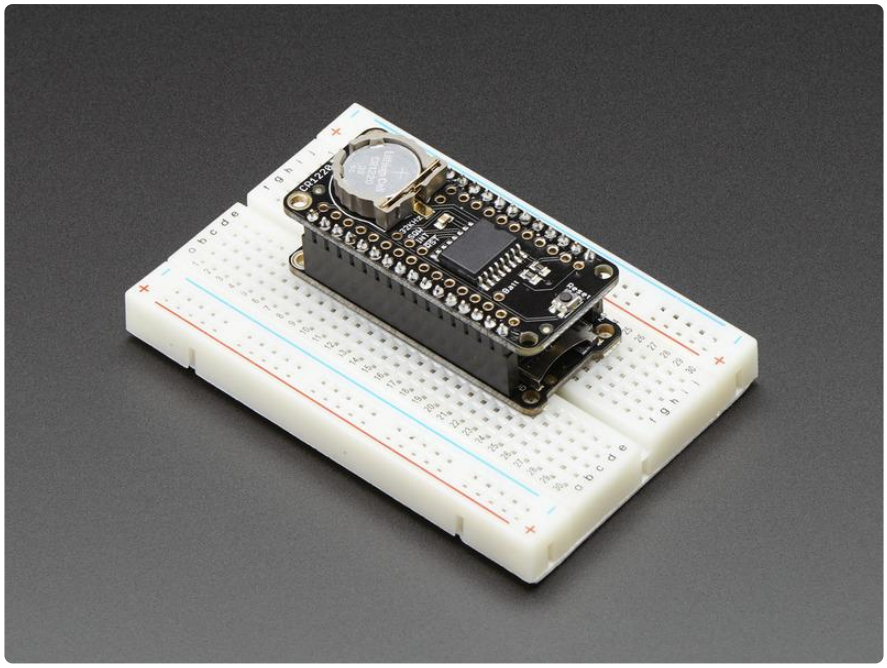


But don't forget, you can also put the stacking headers on the wing and stack the Feather on top of it!

Wiring & Test

Being that the DS3231 is an I2C real time clock, it works with any and all Feathers using the SDA/SCL lines and can share those pins with other I2C devices that do not have the same I2C device

The DS3231 has a default I2C address of 0x68 and cannot be changed

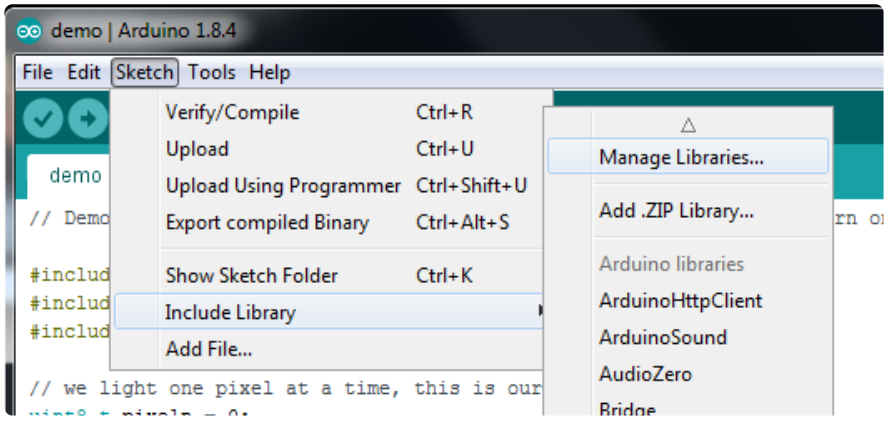


Download RTCLib

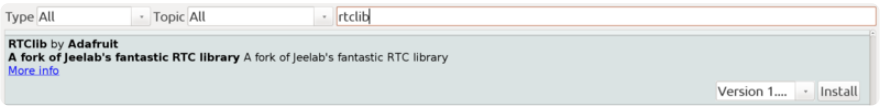
For the RTC library, we'll be using a fork of JeeLab's excellent RTC library RTClib (https://adafru.it/aX2)- a library for getting and setting time from an RTC (originally written by JeeLab, our version is slightly different so please only use ours to make sure its compatible!)

To begin reading data, you will need to download Adafruit's RTCLib from the Arduino library manager.

Open up the Arduino library manager:



Search for the RTCLib library and install it



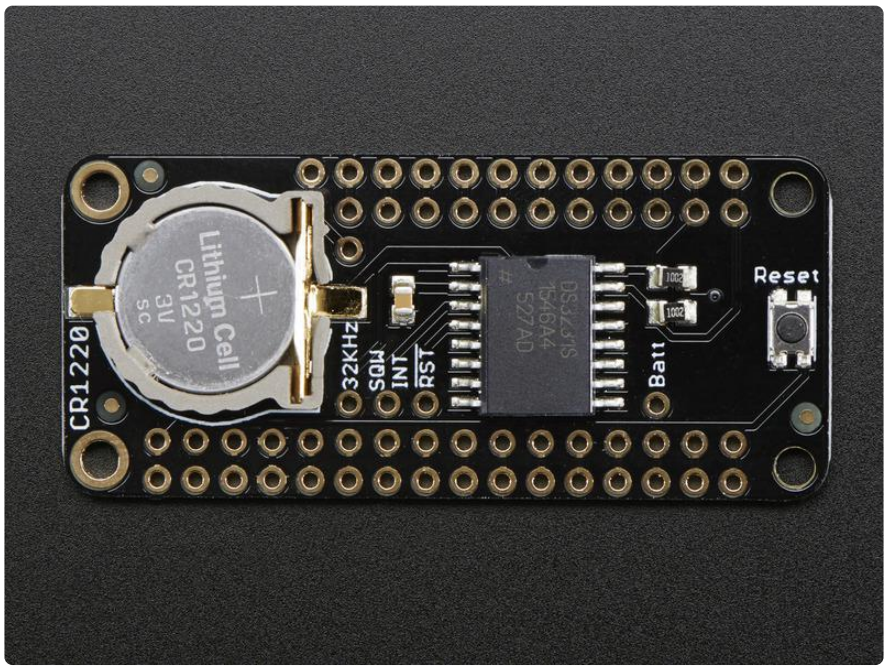
We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

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.

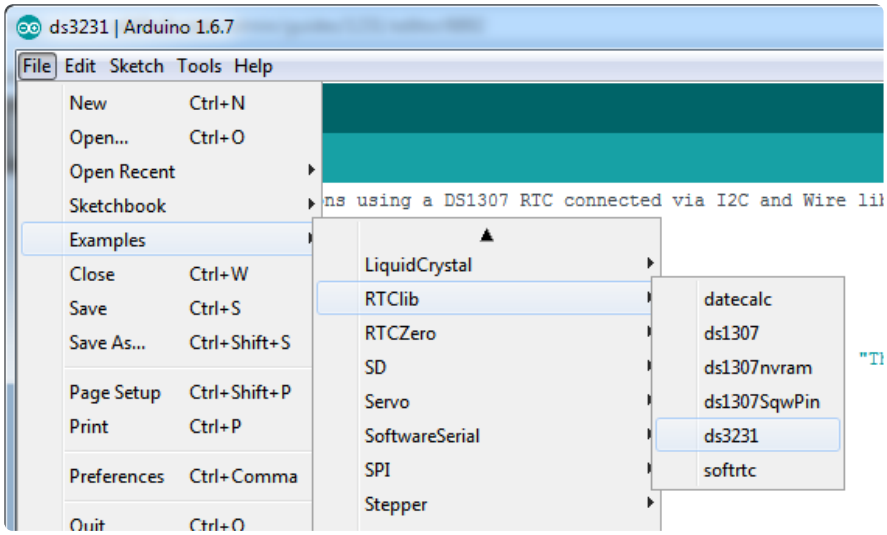
So to start, remove the battery from the holder while the Feather is not powered from a lipoly battery or plugged into USB.

Wait 3 seconds and then replace the battery. This resets the RTC chip.

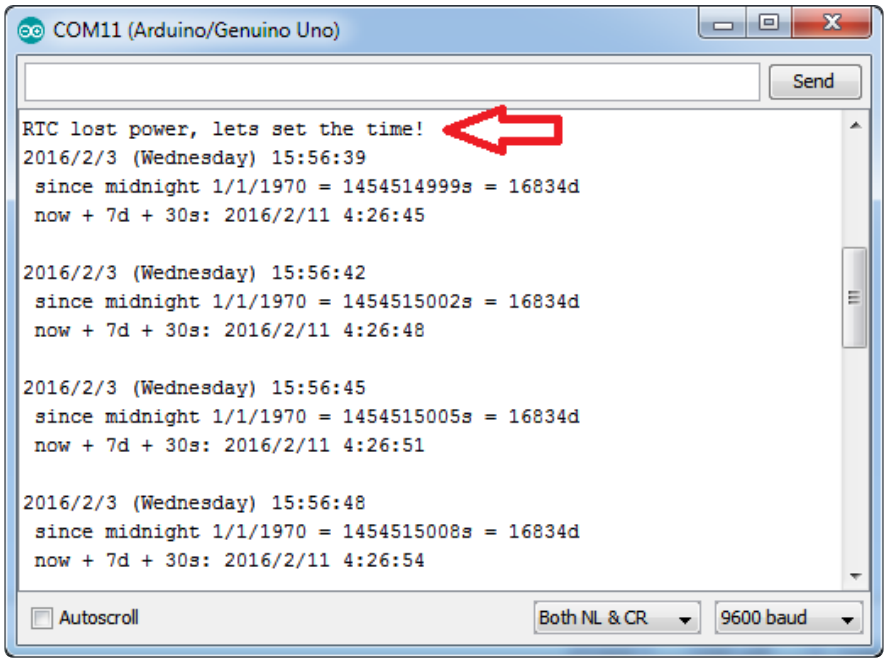


Load Demo

Open up File->Examples->RTClib->ds3231 and upload to your Feather wired up to the RTC

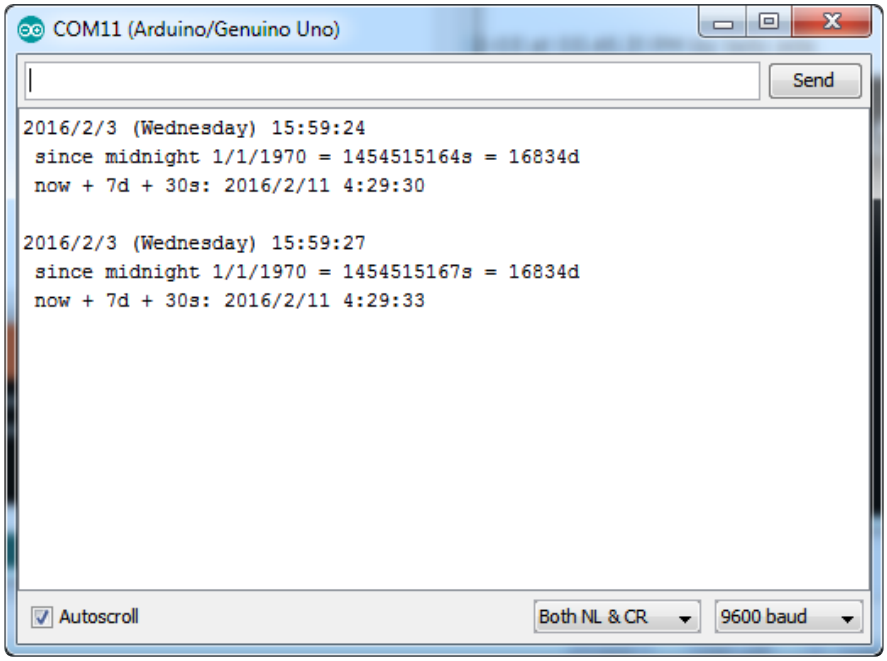


Upload to your Feather and check the serial console @ 9600 baud. After a few seconds, you'll see the report that the Feather noticed this is the first time the DS3231 has been powered up, and will set the time based on the Arduino sketch.



Unplug your Arduino plus RTC for a few seconds (or minutes, or hours, or weeks) and plug back in.

Next time you run it you won't get the same "RTC lost power" message, instead it will come immediately and let you know the correct time!



From now on, you wont 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. Lets 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 no w(), 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. Its 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 midnight 1/1/1970 = ");
    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 its been 5 minutes later, just see if unixtime() has increased by 300, you dont have to worry about hour changes).

Downloads

Datasheets &c

- Maxim product page for the DS3231 (https://adafru.it/EO9)
- Datasheet (https://adafru.it/ldA)
- EagleCAD PCB files on GitHub (https://adafru.it/ohD)
- Fritzing object available in the Adafruit Fritzing Library (https://adafru.it/aP3)

Schematic