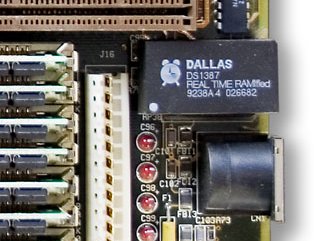
**What is a Real Time Clock?**

When logging data, its 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 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 seperate RTC chip? Well, the biggest reason is that **millis()** only keeps track of time *since the Arduino was last powered* ***-*** that means that when the power is turned on, the millisecond timer is set back to 0. The Arduino 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 on the Arduino? 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 **consistant timekeeping that doesnt reset when the Arduino battery dies or is reprogrammed**. Thus, we include a seperate 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)

[](http://en.wikipedia.org/wiki/File:Realtimeclock_Motherboard_Baby_AT_crop.jpg)  
*This image shows a computer motherboard with a Real Time Clock called the* [DS1387](http://www.maxim-ic.com/app-notes/index.mvp/id/503)*. Theres a lithium battery in there which is why its so big.*

The RTC we'll be using is the [DS1307](http://www.ladyada.net/wiki/partselector/ic#rtc). It's low cost, easy to solder, and can run for years on a very small coin cell.

[](http://www.ladyada.net/images/parts/ds1307.jpg)

As long as it has a coin cell to run it, the DS1307 will merrily tick along for a long time, even when the Arduino loses power, or is reprogrammed.

**You MUST have a coin cell installed for the RTC to work, if there is no coin cell, it will act strangly and possibly hang the Arduino so ALWAYS make SURE there's a battery installed, even if its 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 retreive it. On the Arduino, the pins are 'fixed' to be **Analog 4** and **5** for the built in i2c capability. This is a bit annoying since of course we want to have up to 6 analog inputs to read data and now we've lost two. There is a way to use other pins...however, the hardware is by default set up for **Analog 4** and **5** so lets start with that and then *later* we'll show how to modify the shield!

For the RTC library, we'll be using a fork of JeeLab's excellent RTC library. [Please download it now.](http://www.ladyada.net/make/logshield/download.html) [Then install it in your Arduino directory](http://www.ladyada.net/library/arduino/libraries.html) in a folder called **RTClib**

**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 Arduino is not powered or plugged into USB. Wait 3 seconds and then replace the battery. This resets the RTC chip. Now load up the following sketch (which is also found in **Examples->RTClib->ds1307**) and upload it to your Arduino with the datalogger shield on!

// Date and time functions using a DS1307 RTC connected via I2C and Wire lib

#include <Wire.h>

#include "RTClib.h"

RTC\_DS1307 RTC;

void **setup** () {

**Serial**.begin(57600);

Wire.begin();

RTC.begin();

if (! RTC.isrunning()) {

**Serial**.println("RTC is NOT running!");

// following line sets the RTC to the date & time this sketch was compiled

//RTC.adjust(DateTime(\_\_DATE\_\_, \_\_TIME\_\_));

}

}

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(now.hour(), DEC);

**Serial**.print(':');

**Serial**.print(now.minute(), DEC);

**Serial**.print(':');

**Serial**.print(now.second(), DEC);

**Serial**.println();

**Serial**.print(" since 1970 = ");

**Serial**.print(now.unixtime());

**Serial**.print("s = ");

**Serial**.print(now.unixtime() / 86400L);

**Serial**.println("d");

// calculate a date which is 7 days and 30 seconds into the future

DateTime future (now.get() + 7 \* 86400L + 30);

**Serial**.print(" now + 7d + 30s: ");

**Serial**.print(future.year(), DEC);

**Serial**.print('/');

**Serial**.print(future.month(), DEC);

**Serial**.print('/');

**Serial**.print(future.day(), DEC);

**Serial**.print(' ');

**Serial**.print(future.hour(), DEC);

**Serial**.print(':');

**Serial**.print(future.minute(), DEC);

**Serial**.print(':');

**Serial**.print(future.second(), DEC);

**Serial**.println();

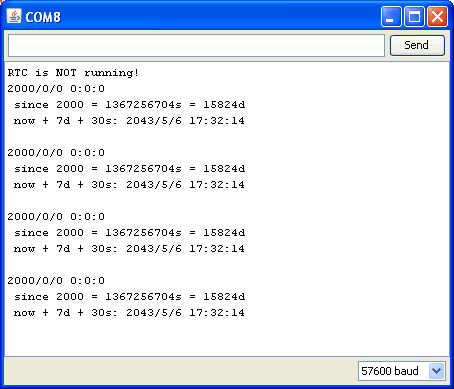
**Serial**.println();

delay(3000);

}

Now run the Serial terminal and make sure the baud rate is set correctly at 57600

bps you should see the following:



Whenever the RTC chip loses all power (including the backup battery) it will report the time as 0:0:0 and it won't count seconds (its stopped). 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 wont 'fall out'

**Setting the time**

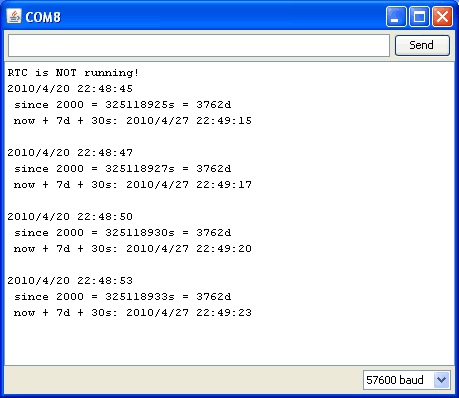
With the same sketch loaded, uncomment the line that starts with **RTC.adjust** like so:

// following line sets the RTC to the date & time this sketch was compiled

RTC.adjust(DateTime(\_\_DATE\_\_, \_\_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 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(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. 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 **get** which counts the number of seconds (not counting leapseconds) since midnight, January 1st 2000

**Serial**.print(" since 2000 = ");

**Serial**.print(now.get());

**Serial**.print("s = ");

**Serial**.print(now.get() / 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 **get()** has increased by 300, you dont have to worry about hour changes)

**'Software' I2C**

Not done! See http://forums.adafruit.com/viewtopic.php?f=25&t=13722

*June 3, 2010 20:55*