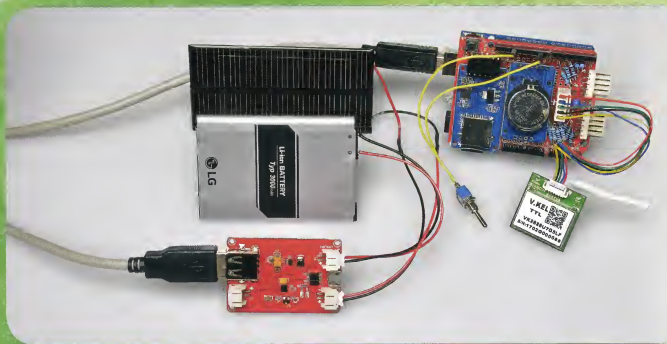


Build An Arduino Data Logger with GPS

This cheap and easy-to-build data logger has four analog and four digital logging channels and can log at intervals from one second to one minute. It runs off a lithium rechargeable cell for an operating time of up to one week (depending upon capacity) and this can be recharged by a small solar cell, so maximum logging time is virtually unlimited. It can also log coordinates from a GPS unit and interface with many different types of sensor.

by Nicholas Vinen

Perhaps its best feature is that it's based on an Arduino with a few low-cost modules attached, so it's easily customisable. Out of the box, it provides support for logging voltages, digital logic states, switch or relay states, temperature, latitude/longitude and frequency (eg, for a flow meter).

If you want to log humidity, barometric pressure, light levels, RF signal strength or just about anything else, you just need to hook up a suitable sensor to the Arduino board and modify the software to read the data off that sensor. Our software will then do the background tasks of power management, saving data to the microSD card and so on.

If you do build this data logger and expand its capability, we hope that you send us the circuit details and revised software so that we can publish it in the Circuit Notebook section of the magazine. That way, others who want to log similar data can do so easily.

Our last data logger project was published in the December 2010, January 2011 & February 2011 issues. That design is now obsolete and we no longer recommend it. Our new design is much easier to set up and we are able to support it with bug fixes, should the need occur. Constructors can easily install updated software using the Arduino IDE and a USB cable.

The old design was also notoriously difficult to interface to a PC, especially if you're using a newer version of Windows than was available at the time (it was designed for Windows 7). The Arduino IDE and drivers are kept up to date for recent operating systems and in fact, since they run on Mac and Linux too, that means this data logger is suitable for a wider audience.

Data logger design

We could have used an Arduino shield specifically intended for data logging which would include an SD card socket, real-time clock and a prototyping area. Instead, we decided to use separate microSD card and real-time clock modules. We had several reasons for this approach.

First, the combination of individual modules costs less, even if you take into account the separate PCB and headers. Second, we are using the DS3231 real-time clock module which is more accurate than the DS1307 often installed on Arduino data logger shields.

And we have used a higher capacity backup battery that's more readily available (CR2032). Third, we may decide to produce a Micromite-based version of this data logger as well, which would be easier to do with individual modules that aren't specifically tied to the Arduino format.

With that in mind, it wouldn't be hard to modify the software for this project to work with Jaycar's XC4536 data logging shield. For example, should you wish to build it using Jaycar's shield, the pins used by the real-time clock and SD card socket on the Jaycar XC4536 are identical to those we're using here.

So all you'd really have to change would be to swap the DS3231 library for the DS1307; a pretty simple change, but one we'll leave up to the reader.

The DS3231 module we're using for timekeeping was described in detail in a separate article in the El Cheapo Module series, in the October 2016 issue, starting on page 33. You can view that article at www.siliconchip.com.au/Article/10296

Similarly, the microSD card interface module we're using was described on pages 74 and 75 of the January 2017 issue and you can view that article at www.siliconchip.com.au/Article/10510

Having decided to use those two modules, we then decided to use two more modules to round out the design. For the power supply, we're using the Elecrow Mini Solar LiPo Charger module which is described in detail in a separate article in this issue, starting on page 44.

A single-cell Li-ion or LiPo cell is hooked up to this board and provides power to the Arduino via a 5V boost regulator, ensuring it has a steady voltage supply even as the cell discharges. This cell can be charged either from a 5V USB source, such as a computer or mains charger or via a small optional solar panel. That means the data logger can be used in a remote location and left for months at a time; as long as it gets enough sun, it will operate continuously.

The other module we're using is an optional GPS receiver. We're recommending the VK2828U7G5LF which we've used on several occasions previously as it is inexpensive but works well. This is used both to ensure the real-time clock is kept accurate and to log the unit's position.

You could use a different GPS receiver but then you will have to figure out how to modify the connections. Or you can leave it off entirely if you don't need the features it provides; the real-time clock will typically gain or lose less than one second per month without it.

Circuit description

The full circuit of the data logger is shown in Fig.1. For our prototype, most of the components are mounted on a prototyping shield which simply plugs into the Arduino (MOD1).

The four analog inputs are available on CON1, along with a ground pin, and connect to the Arduino's A0-A3 analog input pins via 100kΩ/47kΩ resistive voltage dividers. These allow the Arduino to sense voltages up to 15V and protect it from damage from even higher voltages, up to about 60V or -60V. These set the analog input impedance to around 147kΩ.

The digital inputs are on a similar header, CON2 and again, a ground pin is provided. These feed through to digital input pins D2-D5 via 1kΩ series resistors. These are to protect the Arduino from voltage spikes, or voltages outside the range of 0-5V (up to approximately ±15V). Each of these inputs has an internal pull-up current so they will be high if unterminated.

As a result, the digital inputs can be used to sense the presence of a voltage (as long as it is at least 3V) or the state of a switch or relay contact, by connecting one end to the input and the other end to ground. They can also be used to count pulses, for example, from a flow meter, up to about 10kHz.

Digital inputs D0 and D1 of 1 are not used because these are also used as the serial transmit and receive pins for the console. The serial console can

Features and Specifications

Power supply: single Li-ion/LiPo cell with solar charging or 4.5-5.5V USB source (eg, computer or mains charger)

Supply current: average ~30mA; peak ~100mA (with GPS fitted), ~50mA (without GPS)

Battery life: around four days with recommended cell (3Ah); larger capacities can be used

Analog inputs: 4 x 0-15V; protected up to ±60V (maximum voltage can be increased up to 60V)

Digital inputs: four, compatible with 3.3V/5V logic or contact closure; protected up to ±15V

Other inputs: optional GPS lat/lon logging plus 10kHz frequency counter and/or digital temperature sensor. Other sensors (I^2C etc) can be used with software changes

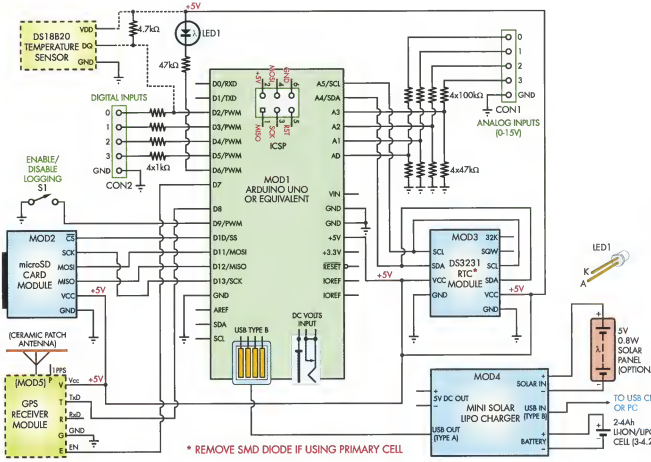
Accuracy: analog inputs ±1% typical with supply voltage calibration; frequency input ±2% typical

Logging interval: defaults to six seconds between entries; 1-60 seconds range is possible

Logging medium: CSV (comma separated value) format text files written to microSD card, up to at least 32GB

Timekeeping: DS3231 real-time clock with battery backup, giving less than one second drift per month

Other features: RAM buffering to reduce power draw; automatic time updates from GPS; logged data can be downloaded via USB serial interface



ARDUINO DATA LOGGER

Fig.1: complete circuit for the Arduino Data Logger, including the optional GPS unit and DS18B20 digital temperature sensor. The rest of the circuit is comprised mainly of modules, such as the Arduino Uno, DS3231 real-time clock module, microSD card interface module and Mini Solar LiPo Charger board.

be used to load data from the unit, via the USB port of a PC, avoiding the need to physically remove the microSD card.

Digital pin D6 is set as an output and drives blue LED1 via a 47kΩ current-limiting resistor. This prevents it from drawing very much current (only about 0.1mA) but it's only lit for a very brief period anyway, so the actual drain on the battery from driving it is almost nothing.

GPS receiver interface

Digital pins D7 and D8 are used to interface with the optional GPS receiver. We're recommending the VK2828U7G5LF as it's a good performer for the price. Keep in mind, that it has an inbuilt ceramic patch antenna so if you are operating indoors, you might get better results using a comparable unit with an external antenna. Having said that, the VK2828 works fine in typical indoor locations.

D7 is used to drive the module's enable (EN) pin; it's held actively low to keep the unit in standby most of the time, resulting in a microamp-level current drain on the battery. Periodically, at a programmable interval that defaults to one hour, the Arduino will bring this pin high to switch on the GPS unit until it gets a lock (usually after about 30 seconds) or if there's insufficient signal, until a timeout occurs (by default, after five minutes).

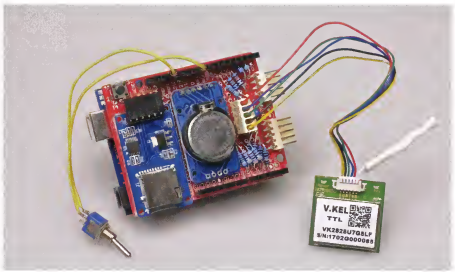
The GPS module draws around 30mA during the time it's powered up; if we assume the average time will be 45 seconds every hour, that works out to 30mA × 45 ÷ 3600 = 375µA average. That's just 0.375 × 24 = 9mAh per day.

Data from the GPS module appears at its TX pin (pin 3) and this is fed to digital input D8. It needs to go to this pin; we explain why below, when describing the operation of the software. The GPS module's RX pin is left unterminated (it has an internal pull-up) as there's no need to send any data to the module. We simply decode its "GPGGA" and "GPRMS" NMEA messages which are sent out by default once per second, at 9600 baud.

The micro can detect whether a GPS module is present based on activity on the D8 pin, or lack of it. D8 has an internal pull-up enabled so that if there is no GPS module connected, it will simply sit high and so the unit will not log GPS co-ordinates.

If a GPS module is detected and is giving sensible output, the latitude, longitude and number of satellites visible will be logged with each entry, along with the number of seconds since a good lock was achieved.

If the GPS module fails to achieve a lock during its power-on period (ie, it times out), the last valid set of readings will continue to be logged and the number of seconds since lock will continue to increase, indicating how "fresh" or "stale" the data is.



The VK2828U7G5LF GPS module shown is an optional extra, if you want to log the unit's location or for greater accuracy in timekeeping, as without it there will be about ±1s of drift in the clock per month.

Digital pin D9 is set as an input, again with an internal pull-up, and connected to external switch S1, which is used to enable or disable logging. This is useful if you want to remove the microSD card to off-load some data; you can simply flick S1 to the off position (where it pulls D9 down to 0V) and the unit will flush any data in its RAM buffer to the SD card and then flash LED1.

You can then remove the card, off-load the data, plug it back in and switch S1 back on to re-enable logging. Or you can simply swap the microSD card for another card to minimise the time without logging.

D9 can also be used where you have a situation where you may only want to log data some of the time. You just need to have it to be pulled low when you don't want to log data, and pulled high or left floating when you do. This can be done with an external relay, switch, microswitch, discrete logic, another microcontroller etc.

SD card interface

Digital pins D10-D13 are wired to the microSD card module and used to read data from and write data to the card. Pins D11, D12 and D13 are hard-wired to the SPI (serial peripheral interface) communication pins MOSI, MISO and SCK on the Arduino respectively. MOSI stands for "Master Out, Slave In", MISO for "Master In, Slave Out" and SCK for "Serial Clock".

While D10 is designated as SS, the hardware Slave Select pin for the SPI bus, in actual fact it is not used by hardware in master mode so we could have used any pin.

But it's conveniently next to the other three so we connect this to the CS/SS (Chip Select/Slave Select) pin on the microSD card module. The only other two connections on that module are to 5V and ground. It has an onboard 3.3V regulator and level shifting circuitry.

The DS3231 real-time clock and calendar module (MOD3) allows the Arduino to keep accurate track of time for time-stamping the log entries, even if power is lost. That module has an onboard battery backup that will last several years and its timekeeping accuracy is very good, at around ±1ppm or about one second per month.

This module has 32kHz and square wave (SQW) outputs which we are not using. We're just connecting the module to a source of 5V power and the Arduino's I^2C serial interface which is hard-coded to analog pins A4 and A5 (unfortunately, limiting us to four analog inputs if we want to use I^2C).

These two pins are enough to allow us to set and query the time and date from the real-time clock module.

That just leaves the battery-backed power supply which is provided by the off-board Elecrow Mini LiPo Charger module. This connects to the Arduino via a standard USB cable, terminated in whatever connector your Arduino module requires; in the case of an Uno, it's a full-size Type B (square) plug, but some Arduino clones use a mini or micro Type B connector instead.

The Charger module can connect to your PC, or a USB charger, via a standard microUSB cable. When connected, it will pass through power to the Arduino but it will also charge the connected Li-ion or LiPo cell from the USB supply. Then, when USB power is removed for whatever reason (whether it's unplugged, or a blackout etc), it will run the Arduino from that cell.

The charger module that can be used with the Data Logger lets a small 5V solar panel be connected in conjunction with a Li-ion/LiPo cell, powering the module and charging the cell. The charger module will favour power coming from the micro-USB port over a cell, meaning you can also have it hooked up to a computer to act as the primary power source, with the cell being a backup.

Note that a Li-ion/LiPo cell has a voltage usually in the range of 3V (flat) to 4.2V (fully charged), while the Arduino expects a steady 4.5-5.5V input. The Elecrow module has an on-board switch-mode step-up regulator to provide this regulated supply. For more details, see our article on LiPo chargers, including that module, elsewhere in this issue.

The Solar Charger naturally also has provision for a solar cell which can run the Arduino and charge the cell in the absence of DC or mains power. We tested our unit with a small 5V, 0.5W solar cell from Oatley Electronics which worked fine.

However, we are recommending that you use a 0.8W cell which we can supply (see Parts List) as it costs about the same and will charge the battery faster, which may be important when the weather is poor.

Software

The software makes use of various Arduino libraries to do all the heavy lifting but even so is quite complex, partly due to the power saving features employed.

We won't fully describe how to utilise and customise the software; but we do detail how to install and run it in the panel "Software Installation". Instead, next month we will have a detailed description of how the software operates. In the meantime, you can download and examine the source code if you already understand C++ software.

Construction

As you will notice from the earlier photos, our unit was built on a protoboard shield and you certainly can do the same. If you're experienced, it will only take you a couple of hours to solder the components onto the shield and complete the point-to-point wiring on the underside to get it all working.

However, it's quite easy to make a mistake when assembling a board this way. So to make it easier and quicker, we've designed a double-sided, shield PCB which you can purchase from our online shop. This comes with a set of stacking headers and costs less than many suppliers charge for a protoboard.

If you do decide to build the unit on a protoshield, note that it's easier if you use 0.25W resistors with thinner leads, since then it's possible to feed two leads into a single hole on the board when wiring up the analog input dividers.

Assuming you're going to take the easier approach and use our custom board, all you really have to do is following the PCB overlay diagram, Fig.2, and the PCB silkscreen to solder each component in place. Start with the resistors, then the right-angle headers, then LED1 (ensuring it's orientated correctly), CON3 and then the two modules.

The DS3231 module normally comes fitted with a right-angle 6-pin header and empty pads at the opposite end. You will therefore need to straighten the right-angle header pins using a pair of pliers and solder a vertical 4-pin header to the other end before soldering the module onto the shield board. This leaves the backup cell on the top, so you can change it easily if necessary.

Note that if you're using the DS3231 with a primary (non-rechargeable) CR2032 cell, you will need to de-solder the small surface mount diode on the board, in a red-tinted glass package. This prevents the module from trying to recharge the non-rechargeable cell.

Having said that, the unit that we recommend you purchase from our website comes with a lithium-ion rechargeable cell so this modification is not required.

We used a 6-pin female header socket with long pins, bent at right angles, to mount our microSD card module on the board. However, we found that this created intermittent problems due to the high-speed nature of the signals carried through these pins. If using this type of socket, at the very least, you should use some M2 machine screws, nuts and spacers to attach the module rigidly to the shield PCB.

However, we feel that a more reliable approach would be to physically mount the module on the shield PCB using either screws and spacers or double-sided tape, then solder rigid wires (eg, from resistor lead off-cuts) between the six pads and the six pins of the module. It won't be removable but, assuming you've made good solder joints, it should operate reliably.

Once all the components have been fitted to the board, it's simply a matter of soldering the four stacking headers in place and then plugging it into the Arduino board. Insert the headers from the top side of the board.

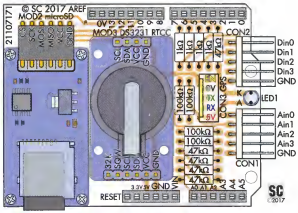
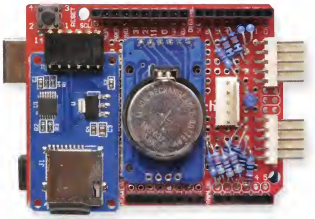


Fig.2: while the Data Logger can be built on a protoshield, it's much easier to use our custom-designed shield PCB. The two main modules, connectors, LED and resistors are fitted to this shield which then plugs into the Arduino board. Refer to the text for our notes about the importance of good connections for MOD2.



When using an Arduino prototyping shield, some of the connections shown in Fig.1 are made by soldering jumper wire between the solder joints on the underside of the shield.

Note that soldering these headers is a little tricky since you need to make the solder joint around the long, protruding pins without getting too much solder on those pins, since they need to plug into the sockets on the Arduino board.

When you do plug the shield in, be careful that the pins go into the right locations on sockets – check the markings on the board. Some Arduino boards have an extra two pins on the lower-left header which can lead to confusion.

If using a GPS receiver, you will need to wire it up to a 5-way polarised header plug to mate with CON3. For the recommended VK2828U7G5LF module, first cut the white (1pps) wire on the supplied cable short, or insulate it (eg, with heatshrink tubing) like we did.

You can then crimp and solder the five remaining wires to the polarised header pins. The colour coding for the wires is shown in the labelling for CON3 in Fig.2. If in doubt, refer to the VK2828U7G5LF data sheet.

When finished, push each pin into the polarised block in the correct location using a very small jeweller's screwdriver or similar implement.

Troubleshooting

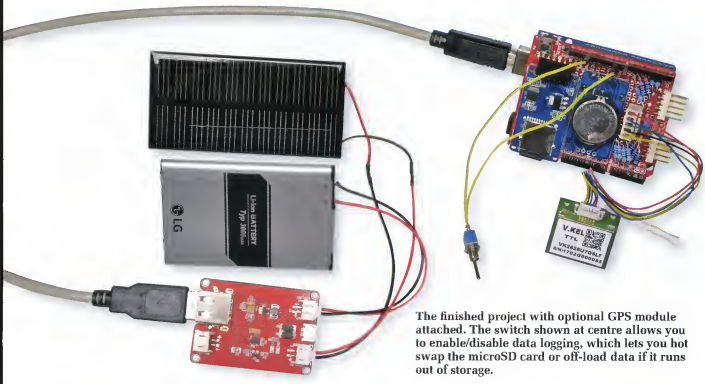
The first thing to do if the data logger isn't working is to plug it into your

Parts List

1 Arduino Uno or equivalent, with suitable USB cable (MOD1) (eg, Jaycar XC4410, Altronics Z6240)
1 double-sided shield PCB, 68.5 x 53.5mm, coded 21107171 (supplied with set of four long pin headers)
OR
1 Arduino prototyping shield (eg, Jaycar XC4482)
1 DS3231-based real-time clock module with backup battery (MOD3) (eg, Silicon Chip online shop Cat SC3519)
1 microSD card module (MOD2) (eg, Silicon Chip online shop Cat SC4019)
1 Elecrow Mini Solar Lipo Charger module with two 2-wire JST 2.0 leads (MOD4) (Silicon Chip online shop Cat SC4308)
1 Li-ion or LiPo cell with built-in protection, capacity around 3Ah (eg, from an old mobile phone or https://hobbyking.com/en_us/turnigy-2000mah-3-7v-w-2-pin-jst-ph.html or similar)
1 microSD card, capacity to suit application
1 5V solar panel of around 0.8W (optional; eg, Silicon Chip online shop Cat SC4339)
1 VK2828U7G5LF GPS module (optional; Silicon Chip online shop Cat SC3362)
1 USB charger with microUSB output (optional, for mains-powered use)
2 5-way right-angle polarised headers (CON1,CON2)
1 6-pin header socket with long pins, 2.54mm pitch (for MOD2)
1 6-pin header, 2.54mm pitch (for MOD3)
1 5-pin polarised header with matching socket (optional; CON3, for GPS module)
1 3mm blue LED
1 SPST or SPDT toggle or slide switch (S1)
1 single male-male jumper lead (for S1)
various length of Kynar (wire wrap wire), ribbon cable strands, light-duty hookup wire or resistor lead off-cuts (if using a protoshield)

Resistors

4 100kΩ	5 47kΩ	4 1kΩ



The finished project with optional GPS module attached. The switch shown at centre allows you to enable/disable data logging, which lets you hot swap the microSD card or off-load data if it runs out of storage.

Software Installation

Once you've finished assembling the unit, download and install the latest Arduino IDE (if you don't have it already). Plug the Arduino main board into your PC and launch the IDE. Before you can upload the sketch, you need to select the port on which the main board is connected. Click on the Tools menu, then Ports and select the right port from the list. It's typically the one at the bottom.

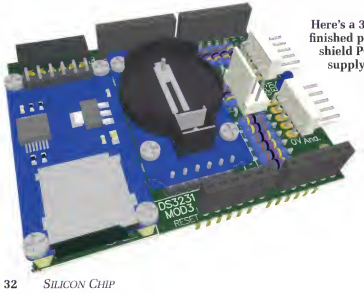
If you haven't already, download the sketch from our website. In the ZIP package, you should find a number of libraries, each of which is also in a ZIP file. Open the Sketch menu in the Arduino IDE, then Include Library and select "Add .ZIP Library". Navigate to the location where you saved the supplied libraries and select the first one. Repeat this process for all the libraries.

You can then open our sketch (using either File→Open or by launching it from your file manager) and select the "Upload" option in the Sketch menu. You should see a progress bar in the lower right corner of the IDE fill from left the right. This will take around 15-30 seconds, depending on the speed of your computer, as it involves compiling the sketch and then uploading it to the Arduino board.

If there are any errors, they will appear in the small window at the bottom of the IDE. The sketch as supplied should compile the first time. If it doesn't, the most likely reason is that you forgot to install one of the libraries, or you already had an incompatible version installed. More likely errors are communications problems, which may suggest that you had wrong port selected. If everything seems OK but it still won't upload, try unplugging and re-plugging the Arduino board and restarting the IDE.

Assuming the upload was successful, you can check the operation of the logger using the Serial Monitor, available under the Tools menu. If you don't see anything in the Serial Monitor, try pressing the reset button on the Arduino board. You should get an output similar to the following:

```
SILICON CHIP Arduino Datalogger powering up
RTC time updated
Calibrating counter
Counter calibration complete (999.30)
Initialising SD card
SILICON CHIP Arduino Datalogger ready
GPS module might be present, checking...
GPS module detected
Opening log file ArduinoLog_2017-06-29_112624.csv
ArduinoLog_2017-06-29_112624.csv
29/06/2017,11:26:24,0.00,0.00,0.00,0.00,1,1,1,0,20.4,1.004,,,,
...
```



Here's a 3D render of the finished project using the shield PCB that we will supply at a later date.

computer and use the Arduino Serial Monitor to look at the debugging messages that it's producing.

Press reset and you should get messages similar to that shown in the adjacent panel. If you get nothing, check that the port setting is correct and try re-uploading the firmware.

Normally, if the firmware gets "stuck", you can tell where based on the last message displayed on the console. If you find it's re-starting repeatedly, or randomly rebooting, the most likely problem is in the connections between the Arduino and the microSD card. Note that the unit will refuse to start up at all if there is no microSD card inserted.

If LED1 is flashing rapidly, this indicates a problem with the RTC module (2Hz) or the microSD card (4Hz).

Note that, because of the buffering, you may not get any logged data output over the serial monitor or written to the microSD card for some time after start-up. If unsure, try changing the state of S1 as this will normally force the unit to flush out any logged data which is buffered.

By default, you will need to wait 36 seconds (6 seconds x 6 buffer entries) after the "Datalogger ready" message to see any logged data.

What happens if the battery goes flat?

If you're powering the unit from a mains USB charger and using the rechargeable cell as a back-up, you shouldn't have to worry about this (unless your area is prone to week-long blackouts!). But if you're using the solar cell, it is possible that a long period of bad weather could result in the cell going flat.

The power supply module does not appear to have a low-battery cut-out feature, which is why we've specified a cell with built-in protection. This will normally prevent it from being over-discharged.

Eventually, the protection circuitry will simply cut power and the logger will shut down, leaving a slightly truncated log file. When power is restored, the cell should begin to charge and the logger should resume operation, opening a new log file.

Fully discharging a lithium-ion/LiPo cell repeatedly can shorten its life but if this happens occasionally, it should not cause any serious problems. SC