

## **Build your own £40 £32 wireless Freezer monitor alarm with a Raspberry Pi. August 2012, revised June 2013**

These instructions will build a radio transmitter you can put inside your -80°C freezer to monitor temperature and warn when the freezer goes over temperature. Everything comes from just 2 places, Ciseco Systems and Amazon. The whole process takes pretty much a whole day to do (once everything's delivered). **UPDATE JUNE 2013** – New “Slice of Radio” adaptor from Ciseco (£9.99) replaces 1 XRF module (£12) and the Slice of Pi Board (£4). Nothing else changes.

### **The components:**

1. Thermistor kit (<http://shop.ciseco.co.uk/temperature-xrf-development-sensor-thermistor/>) £8.50
2. XRF Radio transmitter (<http://shop.ciseco.co.uk/xrf-wireless-rf-radio-uart-rs232-serial-data-module-xbee-shape-arduino-pic-etc/>) £12
3. 1 battery holder for 4 x AA batteries with on-off switch : ([http://www.amazon.co.uk/gp/product/B008SO6XXS/ref=wms\\_ohs\\_product\\_img?ie=UTF8&psc=1](http://www.amazon.co.uk/gp/product/B008SO6XXS/ref=wms_ohs_product_img?ie=UTF8&psc=1)) £2
4. 1 Raspberry Pi, SD card, power supply, etc. – connected to a network (<http://www.raspberrypi.org/>) ~£40
5. Slice of Radio adaptor (<http://shop.ciseco.co.uk/slice-of-radio-wireless-rf-transciever-for-the-raspberry-pi/>) £10
6. 1 Slice of Pi adaptor (<http://shop.ciseco.co.uk/slice-of-pi/>) £4
7. A drinking straw!

If one doesn't wish to use a Raspberry Pi as a dedicated server and already have a Linux server near the Freezer you can use that instead (it needs to run 24/7 and support the Python language), one can simply buy a URF radio transmitter for the USB port (<http://shop.ciseco.co.uk/urf-radio-module-and-serial-inteface-via-usb/>) £18

Minimum costs : Using existing computer, borrowing a soldering iron and doing without the FTDI programming box (i.e. the minimum possible price):  
£8.50 + £12 + £2 + £18 = ~£40

As above but using a Raspberry Pi:  
£8.50 + £12 + £2 + £10 + £4 = £36-50p  
Or with the computer too ~ £80

Theoretically not needed, but in reality very useful, an FTDI box to program the XRF (<http://shop.ciseco.co.uk/ftdi-usb-interface-for-xbee-also-xrf-x232-bluebee-etc/>) £20  
This has now been superseded by the “Explorer plus” box (<http://shop.ciseco.co.uk/explorer-plus-5-in-1-usb-interface-for-xbee-xrf-rfu-etc-exp/>) £22

You might also need to buy a soldering iron ([http://www.amazon.co.uk/Draper-71415-Soldering-Iron-Kit/dp/B000ELJ0C4/ref=sr\\_1\\_1?ie=UTF8&qid=1341764723&sr=8-1](http://www.amazon.co.uk/Draper-71415-Soldering-Iron-Kit/dp/B000ELJ0C4/ref=sr_1_1?ie=UTF8&qid=1341764723&sr=8-1)) £11

The physical build is completely straightforward, the instructions and pictures of the steps are all on the Ciseco website. If you've not done soldering before here's a simple cartoon guide: [http://mightyohm.com/files/soldercomic/FullSolderComic\\_EN.pdf](http://mightyohm.com/files/soldercomic/FullSolderComic_EN.pdf). Do the Slice of Pi board first as it's the easiest, save the fiddly 10K resistor in the thermistor for last.  
The big variation from the Ciseco instructions is to fit a 4 AA battery pack to drive the thermistor sensor rather than the CR2032 button cell supplied, which won't last very long at -80°C. AA Lithium batteries must be used to handle the very low temperatures. The switch is on the left

side of the 4AA battery box. Remove the battery springs and clips from the right hand side of the 4 AA box and move the black wire clip into the second battery slot. Break or cut out the battery separator to clear the right hand side. The thermistor circuit and XRF will fit nicely into this half of the box. The result is a much neater single box setup. 2 x 1.5 v Lithium Ion batteries are used to drive the XRF, they have been shown to last over a year (so far).

### Once it's built:

Assuming that you're using a Raspberry Pi

In the following text the responses from the computer are in red. The instructions are for Debian Linux, the default for the Raspberry Pi. The commands to be typed are preceded by "\$".

### Setting up Debian Linux to use the XRF

You will need to edit 2 files to prevent Linux sending things to the port and free it up for the XRF module to use. The method below is detailed here:

<http://www.irrational.net/2012/04/19/using-the-raspberry-pis-serial-port/>.

An almost identical set of instructions is also available from the Ciseco website:

<http://openmicros.org/index.php/articles/94-ciseco-product-documentation/raspberry-pi/283-setting-up-my-raspberry-pi>

First: Using the nano program, edit the file "cmdline.txt" in the boot directory to not send anything from the system to the serial port (ttyAMA0).

```
$sudo nano /boot/cmdline.txt
```

The cmdline.txt file looks like this (note the strange lack of linefeeds):

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1  
root=/dev/mmcblkop2 rootfstype=ext4 elevator=deadline rootwait
```

and it should be edited down to look like this:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblkop2 rootfstype=ext4 elevator=deadline  
rootwait
```

After closing nano, check the changes have been saved:

```
$ cat /boot/cmdline.txt
```

Second: edit the file "inittab" in the etc directory to not send anything else to the serial port.

```
$ sudo nano /etc/inittab
```

Comment out the following line at the end of the file by inserting a "#" at the start of the line, i.e. change :

```
2:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

to :

```
# 2:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

After closing nano, check the change has been saved:

```
$ cat /etc/inittab
```

Finally reboot the system:

```
$sudo reboot
```

## Setting up the XRF

- 1) Fit the XRF module into the "Slice of Pi" adaptor on the Raspberry Pi. Following the instructions on the Openmicros website ( <http://openmicros.org/index.php/articles/84-xrf-basics/215-xrf-firmware-upload-with-a-raspberry-pi> ), load the XRF ADC analog to digital firmware, called llapanalog-v0.XX-24mhz.bin, where "XX" is the latest version number (50 at the time of writing) into it linked from here: <http://openmicros.org/index.php/articles/84-xrf-basics/180-all-xrf-firmware-updates> .  
NOTE! Don't load the thermistor firmware if you're going for < -40°C ! The thermistor firmware converts the ADC signal into a temperature, but is written for a minimum value of ~-45°C and so gives an error if the temperature goes below that (sending -273).
- 2) Remove the thermistor XRF. Stick the slice of Pi in a drawer somewhere in case you need it again sometime.
- 3) Insert the Slice of Radio into the Raspberry Pi.

## Setting up the software.

These python programs require the pycserial and matplotlib libraries to be installed:

```
$ sudo apt-get install python-serial  
$ sudo apt-get install python-matplotlib.
```

Set up a folder to hold the programs and check they work. Use the "cd" command to change to that directory.

The "Set-cycle" program sets the Thermistor pack to send readings every 30 minutes and so conserves the battery. You may wish to edit the Set-cycle.py file to a more frequent reading whilst you're calibrating the thermistor (see below), then re-run it with the 30 minute setting when you've finished.

```
$sudo python Set-cycle.py
```

**LLAP UTILITY TO SEND CISECO XRF MODULES INTO SLEEP CYCLE,-  
ONLY TESTED FOR THERMISTORS!**

See:

<http://openmicros.org/index.php/articles/87-llap-lightweight-local-automation-protocol/llap-devices-commands-and-instructions/122-xrf-firmware-temp>  
for details (above address should cut and paste).

**NOTE! This program will simply hang if used to set an XRF module that is switched off.  
Use 'ctrl+z' to exit if this happens.**

**This python file is written to set the XRF labelled "--" to cycle. This is the factory default. If the selected device has been given a new ID (by sending "a--CHDEVIDXX", where "XX" is the new name and then sending "a--REBOOT---" to save it), the variable "devid" in this file will need to be edited.**

**The sleep interval is set by a 3 digit code (001 to 999), followed by D (days), H (hours), M (minutes) or S (guess what?). The default is 30 minutes = 030M**

**Once the settings commands are sent, this program sits and listens, displaying the time of each reading received. This allows you to verify that the new cycle is being followed by the XRF.**

**Computer reports time as 15:26 07-12-2012**

Set calibration to 030M

Use use ctrl+z to exit and then edit this file if you want to change this.

Press return key to commit to this setting.

Plug the XRF thermistor module into the thermistor pack and switch it on. Press return to run the cycle loading software. Eventually the program will display the following:

Opening port /dev/ttyAMA0 at 9600 baud.

Monitoring cycle timing and waiting for battery reading at 15:26 07-12-2012

If interval is correct, use ctrl+z to exit

Content = |WAKE---| detected at 16:02 07-12-2012

!!!!!!!!!!!!!!!!!!!!!!!

!Calibration signal sent!

!!!!!!!!!!!!!!!!!!!!!!!

Interval should now change to 030M

Content = |a--BATT2| detected at 16:02 07-12-2012

Content = |.42-| detected at 16:02 07-12-2012

Content = |a--WAKE-| detected at 16:02 07-12-2012

Content = |---a--| detected at 16:02 07-12-2012

Content = |NTVL030M| detected at 16:02 07-12-2012

Content = |a--SLEEP| detected at 16:02 07-12-2012

Content = |ING-| detected at 16:02 07-12-2012

Use 'ctrl+z' to stop the program once the signals have been sent.

The cycle interval can be quite difficult to set twice. The signal can only be sent when the battery and wake readings are being collected, and that's only once every 10 cycles, so IF you're changing the setting on an XRF with a long cycle time, you could be waiting quite a while (for example if the cycle was set to 030M, you could be waiting 5 hours, if you were crazy enough to set it to 040D, it could be over a year!). In practice this setting is easier to change with the thermistor XRF plugged into an FTDI box on a Windows PC.

It is possible that once set up like this the thermistor might run OK using the built in CR2032 battery pack, we've not tried it.

Next check the thermistor calibration:

```
$sudo python temp-monitor.py
```

```
starting...
```

```
string = |a--ANA10541-|
```

```
10541 reading at : Sun Jul 8 18:54:15 2012
```

Leave this running, as it will only report every 30 minutes it might take quite a time to get a reading. Once the first reading has been collected, put the Thermistor pack in a Ziploc bag in the -80C freezer and wait for further readings.

```
string = |a--ANA19741-|
```

```
19741 reading at : Sun Jul 8 19:24:15 2012
```

This reading should be over 19,500. The actual reading is the first 4 digits, the fifth is a minor error in the software (it is in fact it's the last digit of the previous battery reading). Once fully cooled the reading should vary by only the last digit. Different thermistors may have different readouts, you may well need to adjust the Freezer.py software for your individual thermistor. For

our setup, the thermistor gave ~1974 at – 80°C and 1626 at – 21.5°C, so we used (XXXXX-1626)/-4.3 as our conversion to approximate temperature. Another thermistor gave 2023 and 1643, so we had to adjust accordingly.

If the XRFs stop communicating once you've moved the thermistor into the freezer, you may need to boost the signal. Try making a hole in the thermistor box to allow the antenna to be straight (putting a drinking straw over the antenna helps with this). Also the orientation of the 2 antennae should match (both vertical is easiest).

You're almost done!

### Running the Freezer monitor

Edit the "Freezer.py" program to send emails when the temperature is out of range, you'll need to ask your email provider for the answers to these settings:

whereto = 'smtp.xxx.yy.zz' - name of mail server computer

fromaddr = 'aaa@xxx.yy.zz' - email address of sender

toaddr = 'you@xxx.yy. zz' - email address of recipient

You may also need to uncomment the secure server.login lines in the program and set a username and password. HOWEVER, there are clear security implications to having your email details in a plain text file, so you should set up a special email account just for the software (our system allows non-password verified emails to a special server).

The Freezer program prints the readings from the XRF as it collects them. In the first instance, run the program in a window to monitor the output

```
$sudo python Freezer.py
```

#### FREEZER MONITORING PROGRAM.

Data collection program for a thermistor device previously sent into cyclic sleep mode by program "Set-cycle.py". Use "ctrl+z" to stop.

Electrical interference (such as the freezer motor turning on and off) may trigger the Thermistor to send out of cycle, if so a "wrong length message" error is generated. These events can be minimised by moving the antennae and placing the monitoring computer closer to the freezer.

You also often get a buffer-full of readings when the program starts up (the message is an exact multiple of 12 in length). This is normal.

Sometimes the monitor fails to connect properly after the first reading so this screen just stays frozen. The cause is unclear, but the solution is to quit and relaunch. In extremis a full reboot might be needed!

Opening connection and waiting for response...

Startup complete

Freezer monitor initialised : 19:21 07-08-2012

Starting infinite loop, use ctrl+z to quit.

Received |a--ANA19741-| from Thermistor

Temperature is [-77.0] C at :19:24 07-08-2012.

The main output from the program is a plot file ('Freezer.png'), this is updated once an hour and emailed once a week. It can be downloaded or put in a website. Take the thermistor out of the freezer and check that the alarm email messages are sent when it goes out of range.

The program will stop ('hang up') when you close the window or logoff the system. So once you're happy that everything is running as it should, stop the program and re-issue the python command to run continuously using the "nohup" command (there are many ways of doing this in linux, some, such a cron, are superior but more complex to set up).

```
$nohup sudo python Freezer.py &
```

The program should now run continuously, accumulating the last 168 hours (7 days) of freezer temperatures and sending alarms if the freezer gets warm. The "chat" from the freezer monitor is stored in a growing file nohup.out. This needs backing up and deleting periodically. On most systems it would take several years for this file to become big enough to fill the disk (it grows by ~ 3Kb/day)

To stop this background job use the "ps ax" command to identify the job on the system, then "sudo kill -9 XXXX" to stop it, where XXXX is the process ID of the python job (in the unix manual "kill -9" is gloriously described as "Kill with extreme prejudice").

If you wish a file can be put in the "/etc/init.d" folder to automatically restart the software after a power outage. Write a 3 line file like this (note the explicit path to the python file):

```
$sudo nano Freezerstart
```

```
#!/bin/sh
# /etc/init.d/noip
nohup sudo python /home/pi/Desktop/Freezer-alarm/Freezer.py &
```

Save it and then set it to executable using :

```
$sudo chmod 755 /etc/init.d/Freezerstart
```

Finally add it to the startup list using:

```
$ sudo update-rc.d Freezerstart defaults
```

This generates the following warning message:

```
update-rc.d: using dependency based boot sequencing
insserv: warning: script 'Freezerstart' missing LSB tags and overrides
but still works.
```

This will start the software automatically on power up.

## Final thoughts

The threshold for the alarm may need to be adjusted for particular freezer setups (for example freezers which are often opened and closed may exceed threshold quite often and so sending out "false alarms").

The thermistor sends periodic battery readings which are shown on the graph, the batteries should last at least a year and should be changed when the level falls below 2.1V

If you're using python on another computer, for example one with a URF USB stick fitted, the device tty will be different from the Raspberry Pi ('/dev/ttyAMA0'). For example in Mac OSX the URF value is '/dev/tty.usbmodem000001', Once this value has been edited in the python files the programs should run fine.

## Beyond one freezer!

There is no reason that the "Freezer.py" file couldn't be edited to monitor multiple devices (Freezer1.png, Freezer2.png etc.). See the other file on the website (Freezer2.pdf) for details.

## Beyond one computer!

The Freezer.py program is simply listening to the broadcasts from the thermistor, so there's no reason why the software couldn't be run on several computers at once, as long as they're in range. This would of course give insurance against any one computer crashing. Our setup actually now has 3 Raspberry Pis on battery backup power supplies listening to an overlapping set of freezers and fridges in various places.

In the following listings the email addresses etc. are not set. If cutting and pasting this content remember python is exquisitely fussy about spaces and tabs (basically all the indents should be tabs NOT spaces or the program will crash).

The program Set-cycle.py:

```
# SET CYCLE PROGRAM
#
import serial
import datetime
print """
LLAP UTILITY TO SEND CISECO XRF MODULES INTO SLEEP CYCLE,-
ONLY TESTED FOR THERMISTORS!
See:
http://openmicros.org/index.php/articles/87-llap-lightweight-local-automation-protocol/llap-devices-commands-and-instructions/122-xrf-firmware-temp
for details (above address should cut and paste).

NOTE! This program will simply hang if used to set an XRF module that is switched off.
Use 'ctrl+z' to exit if this happens.

This python file is written to set the XRF labelled "--" to cycle. This is the
factory default. If the selected device has been given a new ID (by sending
"a--CHDEVIDXX", where "XX" is the new name and then sending "a--REBOOT----"
to save it), the variable "devid" in this file will need to be edited.

The sleep interval is set by a 3 digit code (001 to 999), followed by D (days),
H (hours), M (minutes) or S (guess what?). The default is 15 minutes = 015M

Once the settings commands are sent, this program sits and listens, displaying
the time of each reading received. This allows you to verify that the new
cycle is being followed by the XRF.

"""
#ID of devid to listen to
devid = '--'
baud = 9600
port = '/dev/ttyAMA0'
interval = "030M"
# 2 readings an hour is enough to allow one to drop out and still get an hourly update
now = datetime.datetime.now()
print "Computer reports time as " + now.strftime("%H:%M %m-%d-%Y")
print "Set calibration to " + interval
print "Use use ctrl+z to exit and then edit this file if you want to change this."
go = raw_input('Press return key to commit to this setting.')
print "Opening port "+ port + " at ", baud, " baud."
ser = serial.Serial(port, baud)
# Clear out anything in the read buffer
ser.flushInput()
# if this XRF has never been put into cycle before, this is all you need :
ser.write('a'+devid+'WAKE-----')
ser.write('a'+devid+'INTVL'+interval)
ser.write('a'+devid+'CYCLE-----')
# See if it worked...
```

```

print "Monitoring cycle timing and waiting for battery reading at " +
now.strftime("%H:%M %m-%d-%Y")
print "If interval is correct, use ctrl+z to exit"
while 1 :
# wait for message, if it's a battery reading we've go just 100 ms to send the WAKE call,
# so put all the checking the time and printing of details AFTER the reading/writing
# and make the if statement just one character long!
    if ser.inWaiting() > 0 :
        llapMsg = ser.read(ser.inWaiting())
        if "B" in llapMsg :
            ser.write('a'+devid+'WAKE-----')
            ser.write('a'+devid+'INTVL'+interval)
            ser.write('a'+devid+'CYCLE----')
            print ' '
            print '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
            print '!Calibration signal sent!'
            print '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
            print ' '
            print 'Interval should now change to ' + interval
            print ' '
# Get time from system
    now = datetime.datetime.now()
    print 'Content = |' + llapMsg + '| detected at '+ now.strftime("%H:%M %m-%d-
%Y")

```

The program temp-monitor.py:

```

import serial
import time

#devid to listen to

devid = '--'
baud = 9600
port = '/dev/ttyAMA0'

# ser.timeout = 10
# ser.flushInput()
# ser.flushOutput()
print "starting..."
ser = serial.Serial(port, baud)
# ser.timeout = 10
ser.flushInput()
ser.flushOutput()

while 1:
    if ser.inWaiting() == 12 :
# good string
        llapMsg = ser.read(12)
#llap msg start
        if devid in llapMsg :
#yes it's for us
#view rest of the data bytes...
            print 'string = |' + llapMsg + "|"
            if 'ANA' in llapMsg :
                temp = llapMsg[6:10]
                print "Good " + temp + " reading detected at : " +
time.asctime( time.localtime(time.time()) )
            elif ser.inWaiting() > 0 :
# bad string
                now = datetime.datetime.now()

```

```

        count = ser.inWaiting()
        llapMsg = ser.read(ser.inWaiting())
        print count, ' bytes detected, wrong length string = |' + llapMsg + "| "+
now.strftime("%H:%M %m-%d-%Y")

```

The program Freezer.py:

```

# Freezer Monitor Program
#
# use matplotlib for the figure :
# sudo apt-get install python-matplotlib to install
# first turn off plotting window
import matplotlib as mpl
mpl.use('Agg')
# then pull in the plot library
import matplotlib.pyplot as plt
#
#use pyserial 2.5 for serial port communication
# sudo apt-get install python-pyserial to install
import serial
# import time functions
import datetime
from datetime import date
import time
# use smtplib for sending email alerts`
# and base 64 for mime encoding of plots
import smtplib
import base64
# use os to write plotfile to remote websites (requires RSA keypair encryption to be set
up for root!)
import os
#
#
#                               SETTINGS
#
# Default settings for program; port, baud rate, temperature threshold, number of readings
to store
# Alarm email address and server details
#
# set up serial port for temperature readings
DEVICE = '/dev/ttyAMA0'
BAUD = 9600
# Device identifier, "--" is factory preset default for XRF modules, change as need be.
devid = "--"
# for multiple alarms one would use devid1 = "AA", devid2="BB", devid3 = "CC" etc.
#
# Set alarm threshold to trigger email (default -70C)
ALARM = -70
# Set number of readings to be collected and plotted (default 7 days)
collection = 168
#
# Set SMTP mailing details for Alarm email
whereto = 'smtp.xxx.yyy.zzz'
fromaddr = 'aaa@xxx.yyy.zzz'
toaddr = 'you@xxx.yyy.zzz'
# Credentials (if needed)
# username = '?????'
# password = '?????'
#
#
#                               END OF SETTINGS
#
#
#
#create data graph arrays and define preset variable values
# set x axis values to ascending numbers 1 - collection

```

```

x_series = range(collection)
# Fill temperature (y axis) array with values 10C below ALARM threshold.
y_series = [ALARM-10] * (collection)
#
# for n devices one would declare :
# y_series = [[ALARM-10 for i in range(collection)] for i in range(n)]
# slightly counter-intuitive declaration, this generates an array of n * collection
# elements, where you can refer to the i'th device readings as y_series[i][reading]
# this hasn't been tested!
#
# declare 1 element array for temperature (needed for array addition)
temp = [ALARM-10]
# set battery level string to "?????"
battlevel = "?????"
# end of variables set up
#
#
# Plot sending email function, happily stolen from :
# http://www.tutorialspoint.com/python/python\_sending\_email.htm
def sendplot(date) :
    filename = "Freezer.png"
# Read a file and encode it into base64 format
    fo = open(filename, "rb")
    filecontent = fo.read()
    encodedcontent = base64.b64encode(filecontent) # base64
    marker = "FREEZER-REPORT"
    body = date
# Define the main headers.
    part1 = "From: " + fromaddr + ""
    To: "" + toaddr + ""
    Subject: "" + date + ""
    MIME-Version: 1.0
    Content-Type: multipart/mixed; boundary=%s
    --%s
    "" % (marker, marker)

# Define the message action
    part2 = ""Content-Type: text/plain
    Content-Transfer-Encoding:8bit

    %s
    --%s
    "" % (body,marker)
# Define the attachment section
    part3 = ""Content-Type: multipart/mixed; name=\"%s\"
    Content-Transfer-Encoding:base64
    Content-Disposition: attachment; filename=%s

    %s
    --%s--
    "" %(filename, filename, encodedcontent, marker)
    message = part1 + part2 + part3
    server = smtplib.SMTP(whereto)
    # login required by many smtp servers (but not ours), uncomment if needed
    # server.starttls()
    # server.login(username,password)
    server.sendmail(fromaddr, toaddr, message)
    server.quit()
    print "Sent plot email"

#
# Use this:
# sendplot(msg)
# to call this function.

```

```

#
print """"

FREEZER MONITORING PROGRAM.

Data collection program for a thermistor device previously sent into cyclic
sleep mode by program "Set-cycle.py". Use "ctrl+z" to stop.
Electrical interference (such as the freezer motor turning on and off) may
trigger the Thermistor to send out of cycle, if so a "wrong length message"
error is generated. These events can be minimised by moving the antennae and
placing the monitoring computer closer to the freezer.
You also often get a buffer-full of readings when the program starts up (the
message is an exact multiple of 12 in length). This is normal.

Sometimes the monitor fails to connect properly after the first reading
so this screen just stays frozen. The cause is unclear, but the solution is
to quit and relaunch. In extremis a full reboot might be needed!
""""

print "Opening connection and waiting for response..."
ser = serial.Serial(DEVICE, BAUD)
print "Startup complete"
print " "
# read the time
now = datetime.datetime.now()
msg = 'Freezer monitor initialised : ' + now.strftime("%H:%M %m-%d-%Y")
print msg
#
# deliberately set the time wrong to trigger our first thermometer recording.
thishour = now.hour - 1
# Set the day of the week (Monday = 0, Sunday =6)
z = str(date.weekday(now))
#
# Initial mail sent on startup. Don't use the sendplot function as "Freezer.png"
# doesn't exist yet!
server = smtplib.SMTP(whereto)
#
# secure login required by many smtp servers (but not ours), uncomment if needed
# server.starttls()
# server.login(username,password)
#
server.sendmail(fromaddr, toaddr, msg)
server.quit()
print " "
print "Starting infinite loop, use ctrl+z to quit."
print " "
#end of startup
#
# Start infinite while loop to listen to XRF module
while 1 :
# All XRF module read and write commands should have 12 characters and begin with the
letter "a"
# Wait for message, the 1 second pause seems to improve the reading when several messages
# are arriving in sequence, such as: a--ANA19734-a--AWAKE----a--BATT2.74-a--SLEEPING-
time.sleep(1)
if ser.inWaiting() == 12 :
    llapMsg = ser.read(12)
    # display packet, helps to troubleshoot any errors
    now = datetime.datetime.now()
    print 'Received |'+ llapMsg + '| from Thermistor at ' +
now.strftime("%H:%M %m-%d-%Y")
    if 'a' == llapMsg[0] :
        #llap msg detected

```

```

if devid in llapMsg :
#yes it's for us
#
# this would need rewiting for multiple alarms,
# Something like
# if devid1 in llapMsg :
#     index=0
# elif devid2 in llapMsg :
#     index=1
# elif devid3 in llapMsg :
#     index=2
# (remember the indices for a 3 element array are 0,1,2)
#
# read the time
now = datetime.datetime.now()

#
# Check for ADC reading or battery packet, ignore anything else.
#
# Is it a battery reading?
if 'BATT' in llapMsg :
# Battery reading sent
    print "Battery level is " + llapMsg[7:10] + "V"
    # Save this value for later.
    battlevel = llapMsg[7:10]
# on our setup this seems to come in as a single 48 byte lump (see
below)
#
# Is it an ADC reading?
if 'ANA' in llapMsg :
# reading sent
    # Has an hour passed?
    if thishour != now.hour :
    # yes,- update the plot
        thishour = now.hour
    # convert string to real number/temperature for
plotting
    try:
        # This conversion is a VERY crude guess of
temperature
        # based on a linear interpolation between just 2
readings
        # 19682 at -78.5C and 16264 at -21.5C
        # more calibration readings would be needed to
fix this.
        # with the default an alarm temperature occurs
when the ADC
        # reading falls below 18874, (18874 - 16264)/-43
= -60
        #
        temp = [((float(llapMsg[6:11]) -16264)/-43)]
        # temp is a 1 element array, hence the "[" "]"
except ValueError:
    # if float operation fails, skip bad reading
    print "bad reading"
#
# shuffle the temperature list to the left, discard
oldest data
    y_series = y_series[1:] + temp
    # for multiple devices this would be :
    # y_series[index][0:collection] =
y_series[index][1:]+temp
    # open plot for data
plt.figure

```

```

# set plot label (timestamp)
freezertime = "Last reading : " +
now.strftime("%H:%M %m-%d-%Y") + ", battery = " + battlevel + "V"
# plot x and y values with label
plt.plot(x_series, y_series, label=freezertime)
#add in axes legends and title
plt.xlabel("Time (hours)")
plt.ylabel("Temperature (approx)")
plt.title("Freezer temperatures over the past week")
#set limits of the x and y axes, the extra 5 on the Y
axis allows space for the label
plt.xlim(0,collection)
plt.ylim(min(y_series)-1, max(y_series)+5)
# set legend position
plt.legend(loc="upper left")
# save figure to png graphics file
plt.savefig("Freezer.png")
# close plot
plt.clf()
#
# send the picture to our webserver
# requires rsa keys to have been set up for root.
#
try :
    os.system("scp Freezer.png
you@xxx.yyy.zzz:/home/Freezer.png")
except :
    print "write to remote host failed"
#
# Check for out of range reading.
if temp[0] > ALARM :
# Freezer is over temperature! Send Alarm email
msg = 'FREEZER ALARM! Monitor reports '+
str(round(temp[0],2)) + "C at " + now.strftime("%H:%M %m-%d-%Y") + ". Battery level : " +
battlevel + " V"
    print msg
    try :
        sendplot(msg)
    except :
        print "mail server returned an error"
#
# Is it time for our weekly report? 3 a.m. on a Friday
# Check the day of the week (Monday = 0, Sunday = 6,
i.e. Friday = 4)
if (date.weekday(now) == 4) and (thishour == 3) :
# for daily readings replace with this:
# if (thishour == 12) :
    msg = 'Weekly freezer report : ' +
now.strftime("%H:%M %m-%d-%Y")
    try :
        sendplot(msg)
    except :
        print "mail server returned an error"
# as the plot is only updated once an hour, thishour
# so we don't need to check this hasn't been sent
#
else :
# goes with first "if" statement (if ser.inWaiting() == 12 : )
# read buffer has non-12 length string in it,- empty the buffer!
if ser.inWaiting() > 0 :

```

```

length = ser.inWaiting()
llapMsg = ser.read(ser.inWaiting())
print 'Received following message from Thermistor |'+ llapMsg + '|
length = ',length
# the 48 byte string :
# a--ANA19734-a--AWAKE-----a--BATT2.74-a--SLEEPING-
# is frequently seen
# Is it a 4 x 12 byte Wake cycle message with a battery reading?
if ('BATT' in llapMsg) and (length == 48) :
# Battery reading sent as part of awake cycle
    print "Battery level is " + llapMsg[31:35] + "V"
    # Save the battery reading.
    battlevel = llapMsg[31:35]
ser.flushInput()
#
# for want of a better phrase,- endwhile
# end of program

```

The following is the content of nohup.out from “Freezer.py” once it’s been up and running for a while using the command “nohup sudo python Freezer.py &”. The thermistor has been set to report every 30 minutes using “Set-cycle.py”:

```

Received |a--ANA19743-| from Thermistor at 06:30 07-14-2012
Received |a--ANA19743-| from Thermistor at 07:00 07-14-2012
Received |a--ANA19743-| from Thermistor at 07:30 07-14-2012
Received following message from Thermistor |a--ANA19743-a--AWAKE-----a--BATT2.33-a--
SLEEPING-| length = 48
Battery level is 2.33V
Received |a--ANA19743-| from Thermistor at 08:30 07-14-2012
Received |a--ANA19743-| from Thermistor at 09:00 07-14-2012
Received |a--ANA19743-| from Thermistor at 09:30 07-14-2012
Received |a--ANA19743-| from Thermistor at 10:00 07-14-2012
Received |a--ANA19743-| from Thermistor at 10:30 07-14-2012
Received |a--ANA19743-| from Thermistor at 11:00 07-14-2012
Received |a--ANA19743-| from Thermistor at 11:30 07-14-2012
Received |a--ANA19743-| from Thermistor at 12:00 07-14-2012
Received |a--ANA19743-| from Thermistor at 12:30 07-14-2012
Received following message from Thermistor |a--ANA19733-a--AWAKE-----a--BATT2.32-|
length = 36
Received |a--SLEEPING-| from Thermistor at 13:00 07-14-2012
Received |a--ANA19742-| from Thermistor at 13:30 07-14-2012
Received |a--ANA19732-| from Thermistor at 14:00 07-14-2012
Received |a--ANA19742-| from Thermistor at 14:30 07-14-2012
Received |a--ANA19732-| from Thermistor at 15:00 07-14-2012
Received |a--ANA19742-| from Thermistor at 15:29 07-14-2012
Received |a--ANA19732-| from Thermistor at 15:59 07-14-2012
Received |a--ANA19742-| from Thermistor at 16:29 07-14-2012

```

Note that the reading time can drift a bit, here it moves from 15:00 to 15:29 then 15:59 minutes, which means the “16:29” reading will be the one taken for that hour and the nominal 15:00 and 16:00 values in the graph will actually be taken 89 minutes apart.