

Data Acquisition from Serial Ports

With Python's pyserial module

Dan Walker

September 26, 2018



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

In computing, a serial port is a serial communication physical interface through which information transfers in or out one bit at a time (in contrast to a parallel port). Throughout most of the history of personal computers, data was transferred through serial ports to devices such as modems, terminals and various peripherals.

(ref: Wikipedia)

- ▶ Very basic form of communication. Low power, low CPU.
- ▶ Protocol is called “RS-232” or “RS-485”
- ▶ In common use on scientific equipment, for example NCAS’s laser ceilometer.
- ▶ Usually a 9-pin or 25-pin “D-Sub” port, but sometimes a variety of others
- ▶ no longer common on computers. However, USB->Serial adapters are approx £15.



The Papouch temperature probe

- ▶ Very basic serial RS232 temp probe
- ▶ approx €20
- ▶ Measures temperatures from -55°C to +125°C, 0.1°C resolution
- ▶ Output is in ASCII format in °C, no conversion needed.
- ▶ Port powered so needs no power source
- ▶ Datasheet included in your kit



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Basic connections

<https://pythonhosted.org/pyserial/shortintro.html>

```
#!/usr/bin/python2.7
import serial

ser = serial.Serial(
    port='/dev/ttyUSB0',
    baudrate=9600,
    bytesize=serial.EIGHTBITS,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE
)
```

Figure out how to read and return data from the ser object.

As well as the pyserial shortintro above, you may find

https://pythonhosted.org/pyserial/pyserial_api.html and the Papouch thermometer datasheet useful.



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Basic connections example

- ▶ baudrate, bytesize, parity and stopbits are the most common parameters that need changing



**National Centre for
Atmospheric Science**
NATURAL ENVIRONMENT RESEARCH COUNCIL

Basic connections example

- ▶ baudrate, bytesize, parity and stopbits are the most common parameters that need changing
- ▶ They depend on the serial device in question



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Basic connections example

- ▶ baudrate, bytesize, parity and stopbits are the most common parameters that need changing
- ▶ They depend on the serial device in question
- ▶ The Papouch thermometer 9600 baud, eight bits, no parity checking and one stopbit. ("9600 8N1")
(Same as pyserial's defaults!)



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Basic connections example

- ▶ baudrate, bytesize, parity and stopbits are the most common parameters that need changing
- ▶ They depend on the serial device in question
- ▶ The Papouch thermometer 9600 baud, eight bits, no parity checking and one stopbit. ("9600 8N1") (Same as pyserial's defaults!)
- ▶ For other parameters, see pyserial's API (https://pythonhosted.org/pyserial/pyserial_api.html)



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL


```
#!/usr/bin/python2.7

import serial

ser = serial.Serial(
    port='/dev/ttyUSB0',
)

print(ser.read(size=8)) # "8" here is specific to the Papouch thermometer device

ser.close()
```

```
$ python readserial_basic.py
+025.1C
```



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Basic connections example - 2

- ▶ `/dev/ttyUSB0` is Linux's way of referring to the first USB serial port
- ▶ subsequent ones `/dev/ttyUSB1`, `/dev/ttyUSB2` and so-on
- ▶ Built-in serial ports would be `/dev/ttyS0`, `/dev/ttyS1` etc.
- ▶ Windows machines, the portname will be of the form COM1, COM2, COM3, etc. (USB ports *normally* start at COM3, but not always)
- ▶ Mac OSX machines are different again. It will be something like `/dev/tty.SOMETHING`, e.g. `/dev/tty.PL2303-xxx`
- ▶ You may need to experiment to determine which the USB converter has attached to.
- ▶ More on this later



Why `ser.read(size=8)` ?

If you refer to the Papouch thermometer datasheet 's "Communication Protocol" section, you will see:

```
<sign><3 characters - integer °C>  
<decimal point><1 character - tenths of °C>  
<C><Enter>
```

as a description of the output. In ASCII coding, each character is one byte so each temperature from the thermometer is eight bytes.



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

The datetime module

For the data to be useful you need to add a date and time reading. Python includes a standard module for this, `datetime` (<http://docs.python.org/2/library/datetime.html>)

- ▶ Can process times in a very wide variety of formats
- ▶ Can deal with different timezones (if you ask it to)
- ▶ Other date/time modules are available, but `datetime` is always available.



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Date and Time formats

- ▶ Timezone should be in UTC or TAI in the majority of cases
- ▶ Use an unambiguous format
- ▶ `isoformat()` produces a standard format by default - rarely is that a problem.
- ▶ `strftime()` will do any format you require (e.g. for documents intended to be read by people)

Add date and time to your output.



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Time and Date example

```
#!/usr/bin/python2.7

from datetime import datetime
import serial

ser = serial.Serial(
    port='/dev/ttyUSB0',
    baudrate=9600,
)

print(datetime.utcnow().isoformat(), ser.read(size=8))

ser.close()
```



`datetime.utcnow().isoformat()` is, as you might expect, a command to return the current UTC in ISO format, e.g.:

```
2014-03-06T11:55:43.852953 +025.3C
```

```
print(datetime.utcnow().isoformat(), ser.read(size=8))
```

- ▶ `datetime.utcnow()` call can return in advance of the `ser.read()` call
- ▶ timestamp and the temperature should be as close as possible
- ▶ store the data in a variable and output the variable and the time at the same time

```
datastring = ser.read(size=8)
```

```
print(datetime.utcnow().isoformat(), datastring)
```



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

```
Python 2.7.3 (default, Feb 21 2014, 13:11:38)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from datetime import datetime
>>> dt = datetime.now()
>>> print(dt)
2016-04-12 17:32:38.806353
>>> dt
datetime.datetime(2016, 4, 12, 17, 32, 38, 806353)
>>> print(dt.strftime('%Y-%m-%d %H:%M:%S'))
2016-04-12 17:32:38
>>> print(dt.strftime('%A, %B %d, %Y'))
Tuesday, April 12, 2016
```



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Continuous logging

You are probably going to want your data capture code to run indefinitely, or at least more than once. You should be familiar with flow control and looping constructs from your Intro To Python. Add a loop to your code to continuously log the reading and time. Try and get the same output using `readline()` instead. Why might this be preferable for other instruments?



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Continuous logging example

In most cases, you will need to log more than one data point. A basic modification is fairly simple, using a while loop:

```
#!/usr/bin/python2.7

from datetime import datetime
import serial

ser = serial.Serial(
    port='/dev/ttyUSB0',
    baudrate=9600,
)

while ser.isOpen():
    datastring = ser.read(size=8)
    print(datetime.utcnow().isoformat(), datastring)

ser.close()
```

returns something like:

```
2014-03-06T14:20:28.147494 +023.9C
2014-03-06T14:20:28.849280 +024.0C
2014-03-06T14:20:38.769283 +024.0C
2014-03-06T14:20:48.688270 +024.1C
```



**National Centre for
Atmospheric Science**
NATURAL ENVIRONMENT RESEARCH COUNCIL

Continuous logging with `readline()`

- ▶ The example thermometer *always* returns exactly eight bytes, and so `ser.read(size=8)` is fine.
- ▶ instruments do not always return fixed-length data, and instead separate the readings (or sets of readings) with a special character.
- ▶ Usually newline or carriage return
- ▶ The `pyserial` module provides `readline()` to handle this case.
- ▶ worked differently prior to python v2.6



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

```
#!/usr/bin/python2.7

from datetime import datetime
import serial, io

ser = serial.Serial(
    port='/dev/ttyUSB0',
    baudrate=9600,
)

sio = io.TextIOWrapper(io.BufferedRWPair(ser, ser, 1), encoding='ascii', newline='\r')

while ser.isOpen():
    datastring = sio.readline()
    print(datetime.utcnow().isoformat(), datastring)

ser.close()
```



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Outputting to a file

Usually you will want to output the data to a file rather than the terminal, so, e.g., the data are on disk in case of a fault or similar.

Alter your code to write the data out to a file.



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Outputting to a file example

```
#!/usr/bin/python
'''This version of the readserial program demonstrates
using python to write an output file'''

from datetime import datetime
import serial, io

outfile='/tmp/serial-temperature.tsv'

ser = serial.Serial(
    port='/dev/ttyUSB0',
    baudrate=9600,
)

sio = io.TextIOWrapper(
    io.BufferedRWPair(ser, ser, 1),
    encoding='ascii', newline='\r'
)
```

(cont)



**National Centre for
Atmospheric Science**
NATURAL ENVIRONMENT RESEARCH COUNCIL

```
with open(outfile, 'a') as f: #appends to existing file
    while ser.isOpen():
        datastring = sio.readline()
        #\t is tab; \n is line separator
        f.write(datetime.utcnow().isoformat() + '\t' + datastring + '\n')
        f.flush() #included to force the system to write to disk

ser.close()
```

(see `python/exercises/example_code/ldfsp.py` in your git checkout)



Locating your serial port device

- ▶ Multiple USB->serial devices on one machine may not come back in the same order on reboot
- ▶ You can get multi-port USB->serial devices
- ▶ On Linux, there are alternative ways of addressing the device

```
[user01@unst ~]$ ls -F /dev/serial/by-id/*  
/dev/serial/by-id/usb-Prolific_Technology_Inc._USB-Serial_Controller_D-if00-port0@
```

- ▶ No use if the devices have the same USB ID (e.g. if they are identical)
- ▶ Even devices that look different may have the same USB id if they are the same electronically

```
[user01@unst ~]$ ls -F /dev/serial/by-path/*  
/dev/serial/by-path/pci-0000:00:14.0-usb-0:1:1.0-port0@  
/dev/serial/by-path/pci-0000:00:14.0-usb-0:2:1.0-port0@
```

- ▶ Identifies them by which port they're plugged in to



Finding out serial connection details

Most instruments with serial access will have a (possibly quite short) section in the manual detailing the connection settings. Of course, your instrument might be quite old and the manual lost, in which case: With any luck, the manual will be on their website. Of course they might be out of business, or unwilling or unable to help, in which case:

Use a search engine of your choice

As long as you get the voltage right (usually 3.3v or 5v) you **probably** can't damage your instrument by trying various combinations of serial settings until something works. 9600-8N1 is usually the best place to start



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Assuming you've got some data of the format:

```
2017-02-22T10:00:08.457120 +019.4C  
2017-02-22T10:00:18.438098 +019.4C  
2017-02-22T10:00:28.419100 +019.4C  
2017-02-22T10:00:38.400093 +019.4C  
2017-02-22T10:00:48.381103 +019.3C  
2017-02-22T10:00:58.362099 +019.3C  
2017-02-22T10:01:08.342102 +019.3C
```

...

You wouldn't really be happy with that as an archive file - if you came back to it in a few years (or someone else did, perhaps after you've moved on) there's several items of missing information.



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Converting data from text to Python datatypes.

Before we write a NetCDF file, we must convert the text file to usable data. Our temperature is in a slightly weird format due to the Papouch sensor including the units, so we need functions to convert the string into a number and the time into a Python 'datetime' object.

$$(T_K = T_C + 273.15)$$



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL

Converting data from text to Python datatypes - 2

```
def convert_time(tm):  
    tm = datetime.strptime(tm, "%Y-%m-%dT%H:%M:%S.%f")  
    return tm  
  
def convert_temp(temp):  
    value = temp.strip("+").strip("C").lstrip("0")  
    return float(value) + 273.15
```

strptime is the opposite of strftime that we used earlier.



Python has a module designed for reading in text formats. It's called `csv`, although it also does tab-separated and related structured ASCII formats. We only need the base reader object here. (Other reader objects deal with more complex cases - f.ex. `DictReader`)

Q1. Read your datafile into Python using the `csv` module such that you end up with list object(s) containing floating-point temperature in K and timestamps as Python `datetime` objects.



Reading data in with the csv module -2

```
infile='sample-serial-temperature-2h.tsv'
outfile='sensor-data.nc'
from csv import reader

# Parse the data into python lists
times = []
temps = []

#open infile and read data into lists
with open(infile, 'rb') as tsvfile:
    tsvreader = reader(tsvfile, delimiter='\t')
    for row in tsvreader:
        times.append(convert_time(row[0]))
        temps.append(convert_temp(row[1]))
```

The call to reader returns an iterator so we can iterate over it with a for loop.



We can write the data from the serial logging exercise to a new NetCDF file.

- ▶ Create a Dataset (use the format `NETCDF4_CLASSIC`)
- ▶ Convert your time series to a suitable CF-compliant series
- ▶ Create a suitable Dimension for your time series
- ▶ Create Variable objects for Temp and Time using appropriate units etc.
- ▶ Assign appropriate metadata to the Temp Variable and and the Dataset
- ▶ Add your time series and temp values to the Dataset
- ▶ Close and write your Dataset. Test that it parses correctly eith `ncdump`



Time series

NetCDF using CF conventions stores times as an offset from a base time rather than an absolute time, so we first subtract `base_time`, and then convert the resulting `timedelta` object to an offset in seconds.

```
# Set reference time and convert datetime values to offset values from reference time  
#reference time is the first time in the input data  
base_time = times[0]  
time_values = []  
  
for t in times:  
    value = t - base_time  
    ts = value.total_seconds()  
    time_values.append(ts)  
  
time_units = "seconds since " + base_time.strftime('%Y-%m-%d %H:%M:%S')
```



Create the NetCDF dimensions & variables

```
# Create the output file (NetCDF dataset)
dataset = Dataset(outfile, "w", format='NETCDF4_CLASSIC')

# Create the time dimension - with unlimited length
time_dim = dataset.createDimension("time", None)

# Create the time variable
time_var = dataset.createVariable("time", np.float64, ("time",))
time_var[:] = time_values
time_var.units = time_units
time_var.standard_name = "time"
time_var.calendar = "standard"

# Create the temp variable
temp = dataset.createVariable("temp", np.float32, ("time",))
temp[:] = temps
```



One of the advantages of NetCDF is that it can contain metadata. We'll set a dictionary to contain it so we don't repeat ourselves.

```
# Set the variable attributes
temp.var_id = "temp"
temp.long_name = "Temperature of sensor (K)"
temp.units = "K"
temp.stabdard_name = "air_temperature"

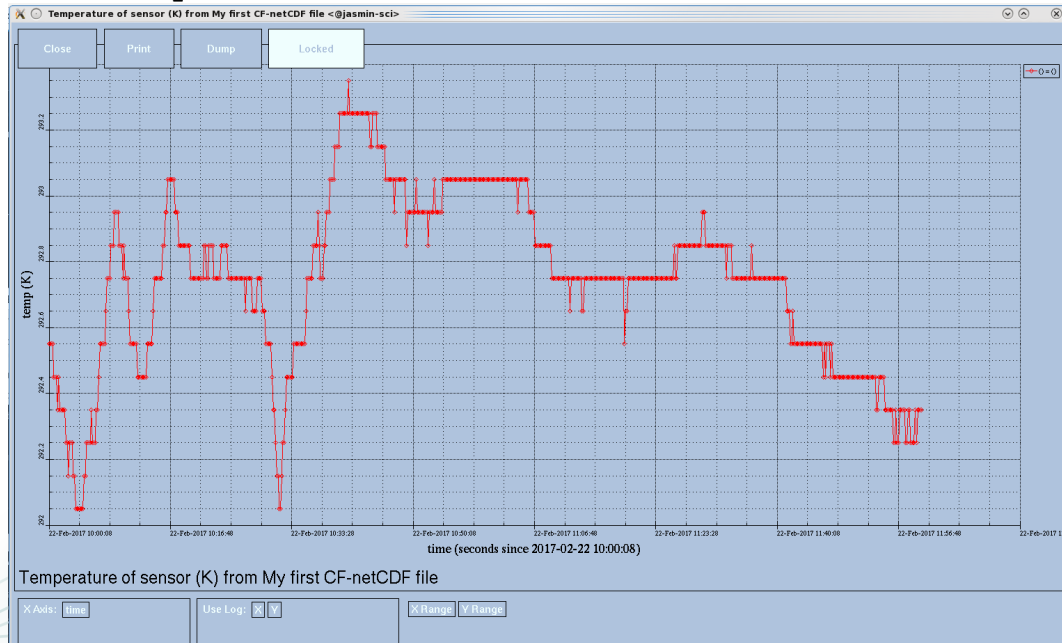
# Set the global attributes
dataset.Conventions = "CF-1.6"
dataset.institution = "NCAS"
dataset.title = "My first CF-netCDF file"
dataset.history = "%s: Written with script: write_sensor_data_to_netcdf.py" % (datetime.now().isoformat())
```



Plotting data

You can do a quick-and-dirty plot with `ncview`:

```
ncview sensor_data.nc
```



Plotting data

```
#!/usr/bin/python2.7

''' Plots a line graph from a NetCDF file '''

from netCDF4 import Dataset
import numpy as np

datafile = 'sensor_data.nc'

nc = Dataset(datafile, mode='r')

temps = nc.variables['temp'][:]
times = nc.variables['time'][:]

times = num2date(times, units=time.units, calendar=time.calendar)

plt.plot_date(times, temps)
plt.savefig('sensor_data.png')
```



Plotting data - with labels

```
After times = num2date(time{:},units=time.units, calendar=time.calendar)
```

```
#get "handles" to affect plot styling
fig, ax = plt.subplots()
#tick every tenth minute
ax.xaxis.set_major_locator(MinuteLocator(byminute=range(0,60,10)))

#format of date on x-axis (display minutes, uses strftime)
ax.xaxis.set_major_formatter(DateFormatter('%H:%M'))

#tick every minute
ax.xaxis.set_minor_locator(MinuteLocator())
ax.autoscale_view()

#line graph
plt.plot_date(times,temps,'-')
labels = ax.get_xticklabels()
plt.setp(labels, rotation=90, fontsize=10, horizontalalignment='center')
plt.xlabel(time.standard_name)
plt.ylabel(temp.standard_name + ' / ' + temp.units)
plt.title(nc.title)

#tidy up layout automatically
fig.tight_layout()
```

Plotting data with CIS (Community Intercomparison Suite)

Another option is CIS

"CIS is an open source command-line tool for easy collocation, visualization, analysis, and comparison of diverse gridded and ungridded datasets used in the atmospheric sciences"

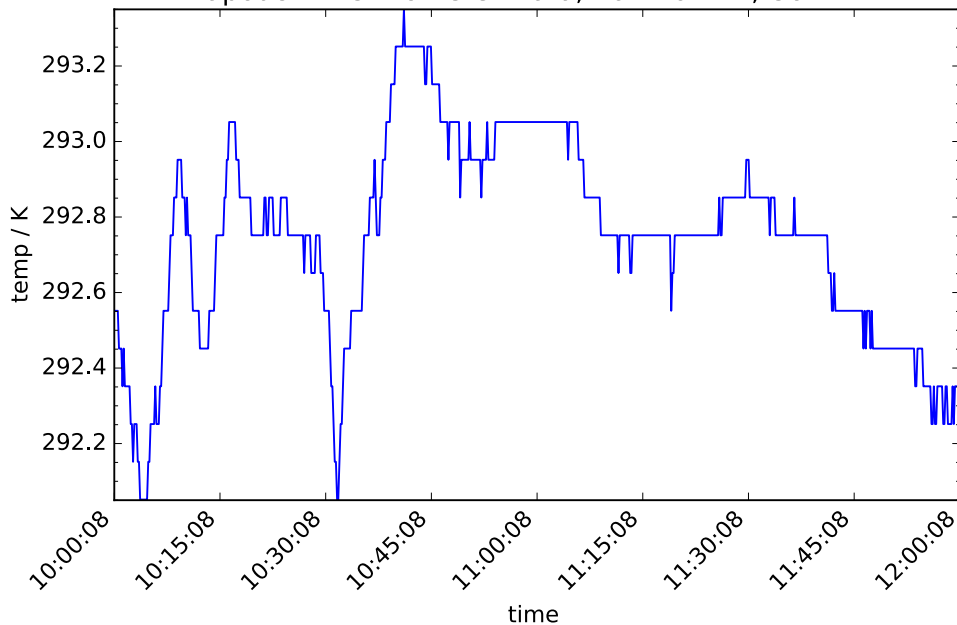
It is based on python. Homepage: <http://www.cistools.net/>

```
cis plot temp:sensor_data.nc --xaxis time --yaxis temp \  
  --title "Papouch Thermometer Data, 2017-02-22, UoL PRD" --xstep "0.010416" \  
  --output sensor_data_sample.svg
```



**National Centre for
Atmospheric Science**
NATURAL ENVIRONMENT RESEARCH COUNCIL

Papouch Thermometer Data, 2017-02-22, UoL PRD



Further exercises

e.g. using `optparse` (for Python < v2.7) or `argparse` (Python \geq 2.7)



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL