NERC SCIENCE OF THE ENVIRONMENT

Logging Data from Serial Ports

With Python's pyserial module

Dan Walker and James Groves

Basic connections

http://pyserial.sourceforge.net/shortintro.html#opening-serial-ports

```
#!/usr/bin/env python

import serial

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

Exercise

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

As well, as the pyserial shortintro above, you may find http://pyserial.sourceforge.net/pyserial_api.html and the Papouch thermometer datasheet useful.

Q1. Why doesn't `readline()` work?

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!

```
#!/usr/bin/env python

import serial

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

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

ser.close()
```

- For other parameters, see pyserial's API (http://pyserial.sourceforge.net/pyserial_api.html)

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

Basic connections example - 2

A note on port names

- `/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)
- You may need to experiment to determine which the USB converter has attached to.

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.

Time and Date

For the read data to be useful you need to add a date and time reading. You can use the module `datetime` (http://docs.python.org/2/library/datetime.html)

Exercise

Add date and time to your output.

Q1. What time format should you use? Why?

Q2. Which timezone?

Q3. Are you sure the datetime call and the reading are at the same time? (within reason)

Time and Date example

```
#!/usr/bin/env python

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
```

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)

```
Python 2.6.6 (r266:84292, Jan 22 2014, 09:42:36)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from datetime import datetime
>>> dt = datetime.now()
>>> print dt
2014-03-12 11:46:47.670578
>>> dt
datetime.datetime(2014, 3, 12, 11, 46, 47, 670578)
>>> print dt.strftime('%Y-%m-%d %H:%M:%S')
2014-03-12 11:46:47
>>> print dt.strftime('%A, %B %d, %Y')
Wednesday, March 12, 2014
```

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.

Exercise

Add a loop to your code to continuously log the reading and time.

Q1. What sort of looping construct are you using? Why?

Q2. What condition causes the loop to exit?

Further exercise if you have time.

Try and get the same output using `readline()` instead. Why might this be preferable for other instruments?

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/env python

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
2014-03-06T14:20:58.608165 +024.1C
```

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

```
#!/usr/bin/env python

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()
```

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. In a modern version of python, it is good practice to use the `with` statement:
http://www.pythonforbeginners.com/files/with-statement-in-python

Exercise

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

Outputting to a file example

```
#!/usr/bin/env python
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')

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

ser.close()
```

Further exercises

Other instruments

Command-line options

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