THE SOFTWARE OF KEYSIGHT U1272A AND MACOS

The software

Obviously it's a pain to keep typing commands into a terminal emulator, so I wrote a toy utility to save my fingers. Perl seemed a good choice because compiling's a bore, and nothing here is remotely performance critical. Given that the command set is based on SCPI I'd hoped to find some helpful Perl modules lying around on CPAN too.

A couple of things seemed as though they might help: GPIB and Lab::Instrument. Sadly though, neither actually helped. Both packages are large and quite complicated: GPIB seems to have suffered bitrot and didn't compile, whilst Lab::Instrument wants an underlying C library.

All of this seems rather large and baroque: the SCPI spec runs to nearly 1000 pages, and both of Perl libraries bristle with classes. If you're building a big experiment I can see the sense in this, but I just wanted a simple way to tickle the serial port.

So my toy program is a thin wrapper around Device::SerialPort It's a noddy program, very much in the `send a line, read a line' style, but knows enough to handle error conditions from the meter, and to iterate through the saved data log.

I should emphasize that it's not proper production quality code though: regard it more as research than solution. For example, the code assumes that there's just a single /dev/tty.PL* device and that it corresponds to the Keysight meter.

A more significant limitation is its failure to do much parsing of the data coming back from the meter.

Installation

The only real dependency is Device::SerialPort, which you can get from CPAN. So, once you've installed the device driver all you'll need to do is:

```
$ sudo cpan Device::SerialPort
...
$ curl http://www.mjoldfield.com/atelier/2011-06/u1272a.pl -o u1272a
$ chmod a+rx ./u1272a
$ ./u1272a
```

Basic operation

The basic command queries the meter for some basic data. You'll notice the absence of any parsing!

```
$ ./u1272a
Opened /dev/tty.PL2303-003312FD :)
—
battery: 77%
config: '"V,0,DC"'
identity: 'Keysight Technologies,U1272A,MY12345678,V1.30'
reading: +4.23800000E-02
reading2: +2.09500000E+01
```

Tracing

If you want to watch what's happening at a low-level add the --trace option:

```
$ ./u1272a --trace
Opened /dev/tty.PL2303-003312FD :)
0.000 >: *IDN?
0.070 <: Keysight Technologies,U1272A,MY12345678,V1.30
0.070 >: SYST:BATT?
0.092 <: 77%
```

0.092 >: CONF?

0.110 <: "V,0,DC"

0.110 >: FETC?

0.142 <: +4.23500000E-02

0.142 >: FETC? @2

0.174 <: +2.09500000E+01

—

battery: 77%

config: '"V,0,DC"'

identity: 'Keysight Technologies,U1272A,MY12345678,V1.30'

reading: +4.23500000E-02

reading2: +2.09500000E+01

Downloading logs

Probably the single most useful task for the program is to grab the meter's data logs. The --get_log option does this, but the returned data are parsed very crudely. Please check the output carefully.

Specifically we read the entire log and write it to a text file, one measurement per line. The meter returns a string of digits like "AABBBBBCCCCCC" for each datum, where BBBBB contains the measurement.

This is written to the file as NNNNN BBBBB AA CCCCCC, where NNNNN just counts upwards.

So, to plot the readings get the x-coordinate from the first column, and the y-coordinate from the second. In gnuplot:

```
gnuplot> plot 'auto.txt' using 1:2 with dots
```

Here's an example:

```
$ ./u1272a --get_log=AUTO
Opened /dev/tty.PL2303-003312FD :)
Grabbing AUTO log:
    0: .........
 1000: .........
 2000: .......
Finished
2727 data from AUTO log written to auto.txt
```

Source: http://www.mjoldfield.com/atelier/2011/06/agilent-macos.html