Data Analysis Program Use

Ben Jolitz

Abstract:

To explain the usage of both data analysis programs built on the "**Muon Life Time and Count Experiments Specification**". The secondary objective is to outline the importation of program output into MS Excel and improvement suggestions for such programs.

Introduction:

The balloon.py and muon.py python programs are designed to read text capture data from the QuarkNet DAQ boards and print out human/Excel friendly output. They are also easy to maintain. Because of python class definition, they can be imported as python modules, and index into other programs.

Code:

The code is python, and is known to work with python 2.5.1.

muon.py

```
# Muon.py
# Written by Benjamin T. Jolitz for SCIPP/Quarknet
# The original author must be named in all direct
# derivative programs and must be named in
# this program
# read and translate lines of data

class data :

    rollover_time = 0 # time ticks of rollover overflow
    last_tim = 0      # previous time stamp

    trigger = False          # accumulating a sample?
    trigger_start_time = 0  # starting time for this trigger
    trigger_accumulator = [] # list of data in sample
```

```
   def __init__(self, file) :

      for line in file.readlines() :
         columns = line.strip().split(' ')
#           print columns

         #b - adjust time (in 24ns ticks) by advancing when it exceeds maximum value
         tim = int(columns[0], 16)
         if tim < self.last_tim :
            self.rollover_time += 4294967296

         data_timestamp = 24L * (tim + self.rollover_time) # data time stamp in nanoseconds

         self.last_tim = tim
         #t - print data_timestamp

         if columns[1] == '80' : # trigger
            self.trigger = True
            self.trigger_time = data_timestamp

         if self.trigger :
            self.trigger_accumulator = self.trigger_accumulator + [columns]

         if self.trigger and columns[5] == '00' : # last line of data in sample


            #j - do something with sample - in this case, print it's data lines
            print 'lifetime', data_timestamp - self.trigger_start_time
#D@--#               print self.trigger_accumulator

            # clear trigger and accumulated data
            self.trigger = False
            self.trigger_accumulator = []



# locate and open file, read and translate time stamped data


from sys import argv, exit

if len(argv) <> 2 :
   print '%s: usage: %s datafile' % (argv[0], argv[0])
   exit(1)

datafile = open (argv[1], 'r')

data(datafile)
```

Balloon.py – Proven to work with ADOM data. However, in case of error, it is most likely some illegal line that splits into 16 pieces. In my experience, the dropouts leading up to the error are key clues, and looking for special unicode

characters (smileys) always find illegal formations.

```
# Balloon.py
# Written by Benjamin T. Jolitz for SCIPP/Quarknet
# The original author must be named in all direct
# derivative programs and must be named in
# this program
# read and translate lines of data

class data :

   start_time = 0                  # beginning time of experiment (in ns)
   rollover_time = 0                 # time ticks of rollover overflow
   last_tim = 0                   # previous time stamp

   interval_time = 2*60*1000*1000*1000 # interval (in ns)
   interval_count = 1                # number of intervals
   interval_first_line = 0           # first line of interval

   trigger_count = 0                 # number of triggers accumulating in the current interval
   trigger_next_interval = 0           # starting time for this trigger



   def __init__(self, file, ignore_start) :

      line_count = 0
      ignore_start = ignore_start * 1000 * 1000 * 1000 # convert seconds to ns
      seen_start = 0


      for line in file.readlines() :
         line_count = line_count + 1

         columns = line.strip().split(' ')

#           print len(columns)
         if  len(columns) <> 16:
            print 'data dropout, line %d, line reads: ' % line_count, columns
            continue

         #b - adjust time (in 24ns ticks) by advancing when it exceeds maximum value
         tim = int(columns[0], 16)
         if tim < self.last_tim :
            self.rollover_time += 4294967296

         data_timestamp = 24L * (tim + self.rollover_time) # data time stamp in nanoseconds
         self.last_tim = tim

         if self.start_time == 0 :
            #t - first interval
            self.start_time = data_timestamp + ignore_start # beginning of first interval
            self.trigger_next_interval = self.start_time + self.interval_time

         #j - ignore data before the first sample
         if data_timestamp < self.start_time :
            continue
```

```
        else :
            if seen_start == 0 :
                seen_start = 1
                self.interval_first_line = line_count
                if ignore_start <> 0 :
                    print 'ignored lines [1-%d] before interval 1' % (line_count - 1)

        #print data_timestamp

        # have we completed an interval? then print a report and reset for the next interval
        if data_timestamp >= self.trigger_next_interval :
            print "interval %04d: triggers %04d [lines %d-%d]" % (
                self.interval_count, self.trigger_count, self.interval_first_line, line_count-1)
            self.interval_first_line = line_count
            self.interval_count = self.interval_count + 1
            self.trigger_count = 0
            self.trigger_next_interval = self.trigger_next_interval + self.interval_time

        # does this line have a trigger?
        if columns[1] == '80' : # trigger
            self.trigger_count = self.trigger_count + 1

    print 'total lines in file:', line_count



# locate and open file, read and translate time stamped data


from sys import argv, exit

# find what we are to do with what
if len(argv) in [2, 3] :
    start = 0
    if len(argv) == 3 :
        start = int(argv[2])

else :
    print '%s: usage: %s datafile [ omit-first-seconds ]' % (argv[0], argv[0])
    exit(1)

datafile = open (argv[1], 'r')

data(datafile, start)
```

Use:

This section assumes you have a data file. If you do not, please consult the ADOM report or Muon Lifetime report. To use on Windows (all), Linux, BSD, or OSX, make sure you have the Python language installed, which can be downloaded for free at *http://www.python.org*.

datafile is a relative term. In reality, it can be any file name, with the extension.
startminstoignore is minutes of time.
number is the current counter.
triggernum is how many triggers were in that interval
startline# and *endline#* are the ranges for the intervals in line counts.
line# is the line at which data was dropped out.
invalidoutput is the data that was invalid.

interval *number:* triggers *triggernum* [line *startline# endline#*]
data dropout, line *line#*, line reads: ['*invalidoutput*']

Procedure:

1. Open up *Terminal* or *Command Prompt*.
2. Change Directory to where the Python programs are, either *muon.py* and *balloon.py*.
3. Place your data file into the same directory using Explorer, Nautilus, Konqueror, or Finder.

4. Type into the Terminal/Prompt: *python balloon.py datafile startminstoignore* or *python muon.py datafile*
5. The screen will fill up with output. Copy/Paste the output into a text file and import into Excel.

Example:

The following example occurred in Linux. The user is ben, and the computer is terranova.

```
ben@terranova:~$ python balloon.py Desktop/ADOM\ RUN\ 1f
interval 0001: triggers 0337 [lines 1-1089]
interval 0002: triggers 0362 [lines 1090-2358]
interval 0003: triggers 0424 [lines 2359-3681]
interval 0004: triggers 0481 [lines 3682-5305]
interval 0005: triggers 0437 [lines 5306-6769]
interval 0006: triggers 0602 [lines 6770-8790]
interval 0007: triggers 0550 [lines 8791-10717]
interval 0008: triggers 0338 [lines 10718-11881]
interval 0009: triggers 0296 [lines 11882-12939]
interval 0010: triggers 0512 [lines 12940-14620]
interval 0011: triggers 0651 [lines 14621-16706]
interval 0012: triggers 0755 [lines 16707-19197]
interval 0013: triggers 0612 [lines 19198-21176]
interval 0014: triggers 0651 [lines 21177-23234]
data dropout, line 24995, line reads:  ['DS']
data dropout, line 24996, line reads:  ['']
data dropout, line 24997, line reads:  ['DS', 'S0=00330293', 'S1=000E13DE', 'S2=00000000',
'S3=00000000', 'S4=0000535A', 'S5=00000000']
data dropout, line 24998, line reads:  ['']
interval 0015: triggers 0492 [lines 23235-25011]
data dropout, line 25032, line reads:  ['CD']
data dropout, line 25033, line reads:  ['']
data dropout, line 25039, line reads:  ['30', 'MIN', 'RUN', 'AT', 'GROUND,', 'NOT', 'BALLOO']
data dropout, line 25040, line reads:  ['Command', 'Error']
data dropout, line 25041, line reads:  ['N', 'FLIGHT']
data dropout, line 25042, line reads:  ['']
data dropout, line 25043, line reads:  ['Command', 'Error']
```

```
total lines in file: 25043
ben@terranova:~$
```

Excel import:

Excel import is very easy. Open MSExcel (or equivalent spreadsheet program), and hit “Open File”. Change “File Type” to “All Files”. Find the directory of your output text file, and open it. It will pop up a wizard, at which you adjust to satisfaction.

Suggestions:

The data analysis program needs a GUI, preferably written in *Tkinter* or setup as a *ncurses*-based CUI.

The *muon.py* program needs to be expanded and retested. As of now, the logic is mostly complete, but it needs testing.

Conclusion:

This program is very easy to use, and conveniently will produce usable data from large text files, instead of importing into Excel and jamming the system. It is very efficient, storing all read data in memory, using speed of the memory, which is always faster than disk.