

---

# Muonic

Jan 11, 2023



---

## Contents:

---

<b>1</b>	<b>muonic - a python gui for QNET experiments</b>	<b>1</b>
1.1	Licence and terms of agreement . . . . .	1
<b>2</b>	<b>muonic setup and installation</b>	<b>3</b>
2.1	prerequisites . . . . .	3
2.2	installation with the setup.py script . . . . .	3
2.3	installing muonic without the setup script . . . . .	3
<b>3</b>	<b>How to use muonic 4</b>	<b>5</b>
3.1	Overview . . . . .	5
3.1.1	Data wells . . . . .	5
3.1.2	Data sinks . . . . .	5
3.2	Start muonic4 . . . . .	6
<b>4</b>	<b>How to use muonic (OLD)</b>	<b>7</b>
4.1	start muonic . . . . .	7
4.2	Saving files with muonic . . . . .	8
4.3	Performing measurements with muonic . . . . .	8
<b>5</b>	<b>Fermilab DAQ - hardware documentation</b>	<b>11</b>
5.1	ASCII DAQ output format . . . . .	11
5.2	DAQ onboard documentation . . . . .	11
5.2.1	V1 . . . . .	12
<b>6</b>	<b>rewrite package</b>	<b>15</b>
6.1	Subpackages . . . . .	15
6.1.1	rewrite.lib package . . . . .	15
6.1.1.1	Subpackages . . . . .	15
6.1.1.2	Submodules . . . . .	25
6.1.1.3	rewrite.lib.Skyview module . . . . .	25
6.1.1.4	Module contents . . . . .	26
6.2	Submodules . . . . .	26
6.3	rewrite.example_measurement module . . . . .	26
6.4	rewrite.runRates module . . . . .	26
6.5	rewrite.runServer module . . . . .	26
6.6	rewrite.runWriterToMongoDB module . . . . .	26
6.7	rewrite.simpleClient module . . . . .	27

6.8	Module contents . . . . .	27
<b>7</b>	<b>Indices and tables</b>	<b>29</b>
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>

---

## muonic - a python gui for QNET experiments

---

The muonic project provides an interface to communicate with QuarkNet DAQ cards and to perform simple analysis of the generated data. Its goal is to ensure easy and stable access to the QuarkNet cards and visualize some of the features of the cards. It is meant to be used in school projects, so it should be easy to use even by people who do not have lots LINUX background or experience with scientific software. Automated data taking can be used to ensure no valuable data is lost.

### 1.1 Licence and terms of agreement

Muonic is distributed under the terms of GPL (GNU Public License). With the use of the software you accept the conditions of the GPL. This means also that the authors can not be made responsible for any damage of any kind to hard- or software.



---

## muonic setup and installation

---

Muonic consists of two main parts: 1. the python package *muonic* 2. a python executable

### 2.1 prerequisites

muonic needs the following packages to be installed (list may not be complete!)

- python-scipy
- python-matplotlib
- python-numpy
- python-qt4
- python-serial
- python-future

### 2.2 installation with the setup.py script

Run the following command in the directory where you checked out the source code:

```
python setup.py install
```

This will install the muonic package into your python site-packages directory and also the executables *muonic* and *which\_tty\_daq* to your usr/bin directory. It also generates a new directory in your home dir: *\$HOME/muonic\_data*

The use of python-virtualenv is recommended.

### 2.3 installing muonic without the setup script

You just need the script *./bin/muonic* to the upper directory and rename it to *muonic.py*. You can do this by typing

```
mv bin/muonic muonic.py
```

while being in the muonic main directory.

Afterwards you have to create the folder *muonic\_data* in your home directory.

```
mkdir ~/muonic_data
```



### 3.1 Overview

Munic consist of different parts in the DAQ chain: Data wells and data sinks. The data wells provide/create/acquire data for the DAQ chain, opposed to data sinks which process the data.

#### 3.1.1 Data wells

The job of data wells is to enter data into the muonic DAQ chain. The data can either be acquired from hardware or pre-recorded data. Currently muonic4 provides these data wells:

1. A DAQ Server, which communicates with the DAQ card and takes live measurements
2. A reader from file, which reads pre-ecorded data from a file.
3. A reader from DB, which reads pre-recorded data from a mongoDB instance.

#### 3.1.2 Data sinks

The job of data sinks is to process data generated by a data well. **Data sinks are not exclusive, meaning multiple sinks can run simultaneously and process the same data.** Currently muonic4 provides these data sinks:

1. A writer to a file, which saves incoming data to a file.
2. A writer to DB, which saves incoming data to a mongoDB instance.
3. A simple client, which simply dumps the incoming data to the terminal.
4. A Pulse Analyzer, which analyses and prints the pulse times.
5. A Rate Analyzer, which analyses the count rates.

## 3.2 Start muonic4

In order to run muonic a data sink and at least one data well must be running. To start a data well *one* of the following commands need to be run before starting a data sink:

- `python3 runServer.py`
- `python3 runReaderFromFile.py`
- `python3 runReaderFromMongoDB.py`

After one of the data well has been started one or multiple data sinks can be started. This can be done by these commands:

- `python3 runPulses.py`
- `python3 runRates.py`
- `python3 runWriterToFile.py`
- `python3 runWriterToMongoDB.py`
- `python3 simpleClient.py`

---

## How to use muonic (OLD)

---

### 4.1 start muonic

If you have setup muonic via the provided setup.py script or if you have put the package somewhere in your PYTHON-PATH, simply call from the terminal

```
muonic [OPTIONS] xy
```

where `xy` are two characters which you can choose freely. You will find these two letters occurring in automatically generated files, so that you can identify them.

For help you can call

```
muonic --help
```

which gives you also an overview about the options.

[OPTIONS]

**-s**

**use** the simulation mode of muonic (no real data, **so** no physics behind!). This should only be used for testing.

**-d**

**debug** mode. Use it to generate more log messages on the console.

**-t** sec

**change** the timewindow for the calculation of the rates. If you expect very low rates, **you** might want to increase it. **default** is 5 seconds.

**-p**

**automatically** write a file with pulsetimes in a non hexadecimal representation

**-n**

**suppress** any status messages in the output raw data file, **might** be useful if you want to use muonic for data analysis.

## 4.2 Saving files with muonic

All files which are saved by muonic are ASCII files. The filenames are as follows:

**Warning:** currently all files are saved under \$HOME/muonic\_data. This directory must exist. If you use the provided setup script, it is created automatically

*YYYY-MM-DD\_HH-MM-SS\_TYPE\_MEASUREMENTTIME\_xy*

- *YYYY-MM-DD* is the date of the measurement start
- *HH-MM-SS* is the GMT time of the measurement start
- *MEASUREMENTTIME* if muonic is closed, each file gets its corresponding measurement time (in hours) assigned.
- *xy* the two letters which were specified at the start of muonic
- *TYPE* might be one of the following:
  - *RAW* the raw ASCII output of the DAQ card, this is only saved if the 'Save to file' button is clicked in the 'Daq output' window of muonic
  - *R* is an automatically saved ASCII file which contains the rate measurement data, this can then be used to plot with e.g. gnuplot later on
  - *L* specifies a file with times of registered muon decays. This file is automatically saved if a muon decay measurement is started.
  - *P* stands for a file which contains a non-hex representation of the registered pulses. This file is only saved if the *-p* option is given at the start of muonic

Representation of the pulses:

*(69.15291364, [(0.0, 12.5)], [(2.5, 20.0)], [], [])*

This is a python-tuple which contains the trigger time of the event and four lists with more tuples. The lists represent the channels (0-3 from left to right) and each tuple stands for a leading and a falling edge of a registered pulse. To get the exact time of the pulse start, one has to add the pulse LE and FE times to the trigger time

---

**Note:** For calculation of the LE and FE pulse times a TMC is used. It seems that for some DAQs cards a TMC bin is 1.25 ns wide, although the documentation says something else. The trigger time is calculated using a CPLD which runs in some cards at 25MHz, which gives a binwidth of the CPLD time of 40 ns. Please keep this limited precision in mind when adding CPLD and TMC times.

---

## 4.3 Performing measurements with muonic

For DAQ setup it is recommended to use the 'settings' menu, although everything can also be setup via the command line in the DAQ output window (see below.) Muonic translates the chosen settings to the corresponding DAQ commands and sends them to the DAQ. So if you want to change things like the coincidence time window, you have to issue the corresponding DAQ command in the DAQ output window.

Two menu items are of interest here: \* Channel Configuration: Enable the channels here and set coincidence settings. A veto channel can also be specified. \* .. note:

You have to ensure that the checkboxes **for** the channels you want to use are checked, before you leave this dialogue, otherwise the channel gets deactivated.

**Note:** The coincidence is realized by the DAQ in a way that no specific channels can be given. Instead this is meant as an 'any' condition. So 'twofold' means that 'any two of the enabled channels' must claim signal instead of two specific ones (like 1 and 2).

**Warning:** Measurements at DESY indicated that the veto feature of the DAQ card might not work properly in all cases.

- **Thresholds:** For each channel a threshold (in milliVolts) can be specified. Pulse which are below this threshold are rejected. Use this for electronic noise suppression. One can use for the calibration the rates in the muon rates tab.

**Note:** A proper calibration of the individual channels is the key to a successful measurement!

In the first tab a plot of the measured muon rates is displayed. A trigger rate is only shown if a coincidence condition is set. In the block on the right side of the tab, the average rates are displayed since the measurement start. Below you can find the number of counts for the individual channels. On the bottom right side is also the maximum rate of the measurement. The plot and the shown values can be reset by clicking on 'Restart'. The 'Stop' button can be used to temporarily hold the plot to have a better look at it.

**Note:** You can use the displayed 'max rate' at the right bottom to check if anything with the measurement went wrong.

**Note:** Currently the plot shows only the last 200 seconds. If you want to have a longer time range, you can use the information which is automatically stored in the 'R' file (see above).

A lifetime measurement of muons can be performed here. A histogram of time differences between succeeding pulses in the same channel is shown. It can be fit with an exponential by clicking on 'Fit!'. The fit lifetime is then shown in the above right of the plot, for an estimate on the errors you have to look at the console.

**The measurement can be activated with the checkbox. In the following popup window the measurement has to be configured. It d**

- more than one pulse appears in the single pulse channel and none pulse is measured in the double pulse channel
- one pulse in the single pulse channel appears and exactly two pulses in the double pulse channel.

**Warning:** The error of the fit might be wrong!

In this tab the muon velocity can be measured. The measurement can be started with activating the checkbox. In the following popup window it has to be configured.

**Warning:** The error of the fit might be wrong!

You can have a look at the pulsewidths in this plot. The height of the pulses is lost during the digitization process, so all pulses have the same height here. On the left side is an oscilloscope of the pulsewidths shown and on the right side are the pulsewidths collected in an histogram.

In this tab you can read out the GPS information of the DAQ card. It requires a connected GPS antenna. The information are summarized on the bottom in a text box, from where they can be copied.

The last tab of muonic displays the raw ASCII DAQ data. This can be saved to a file. If the DAQ status messages should be suppressed in that file, the option `-n` should be given at the start of muonic. The edit field can be used to send messages to the DAQ. For an overview over the messages, look here ([link not available yet!](#)). To issue such a command periodically, you can use the button 'Periodic Call'

---

**Note:** The two most important DAQ commands are 'CD' ('counter disable') and 'CE' ('counter enable'). Pulse information is only given out by the DAQ if the counter is set to enabled. All pulse related features may not work properly if the counter is set to disabled.

---

---

Fermilab DAQ - hardware documentation

---

## 5.1 ASCII DAQ output format

sample line of DAQ output - example for the daq data format

trig- gers	r0	f0	r1	f1	r2	f2	r3	f3	onepps	gp- stime	gps- dte	gps- valid	gps- satellites	xx	cor- rec- tion
92328FE00	00	3D	00	3E	00	00	00	00	915E10C034016.0160180V	00	00	00	00	0	+0055

## 5.2 DAQ onboard documentation

Online help on the DAQ cards is available by sending the following commands to the DAQ

- V1, V2, V3
- H1,H2

## 5.2.1 V1

Setting	example value	description
Run Mode	Off	CE (cnt enable), CD (cnt disable )
Ch(s) Enabled	3,2,1,0	Cmd DC Reg C0 using (bits 3-0)
Veto Enable	Off	VE 0 (Off), VE 1 (On)
Veto Select	Ch0	Cmd DC Reg C0 using (bits 7,6)
Coincidence 1-4	1-Fold	Cmd DC Reg C0 using (bits 5,4)
Pipe Line Delay	40 nS	Cmd DT Reg T1=rDelay Reg T2=wDelay 10nS/cnt
Gate Width	100 nS	Cmd DC Reg C2=LowByte Reg C3=HighByte 10nS/cnt
Veto Width	0 nS	Cmd VG (10nS/cnt)
Ch0 Threshold Ch1 Threshold Ch2 Threshold Ch3 Threshold	0.200 vlts 0.200 vlts 0.200 vlts 0.200 vlts	
Test Pulser Vlt Test Pulse Ena	3.000 vlts Off	

Example line for 1 of 4 channels. (Line Drawing, Not to Scale):

```

Input Pulse edges (begin/end) set rising/falling tags bits.
_____~_____ Input Pulse, Gate cycle begins
_____~_____ Delayed Rise Edge 'RE' Tag Bit
_____~_____ Delayed Fall Edge 'FE' Tag Bit
_____~_____ Tag Bits delayed by PipeLnDly
_____|_____ PipeLineDelay : 40nS
_____ Capture Window: 60nS
_____|_____ Gate Width : 100nS

```

If 'RE', 'FE' are outside Capture Window, data tag bit(s) will be missing.

CaptureWindow = GateWidth - PipeLineDelay

The default Pipe Line Delay is 40nS, default Gate Width is 100nS.

Setup CMD sequence for Pipeline Delay. CD, WT 1 0, WT 2 nn (10nS/cnt)

Setup CMD sequence for Gate Width. CD, WC 2 nn(10nS/cnt), WC 3 nn (2.56uS/cnt)

```

H2
Barometer      Qnet Help Page 2
BA             - Display Barometer trim setting in mVolts and pressure as mBar.
BA d           - Calibrate Barometer by adj. trim DAC ch in mVlts (0-4095mV).
Flash
FL p           - Load Flash with Altera binary file(*.rbf), p=password.
FR             - Read FPGA setup flash, display sumcheck.
FMR p          - Read page 0-3FF(h), (264 bytes/page)
Page 100h= start fpga *.rbf file, page 0=saved setup.
GPS
NA 0           - Append NMEA GPS data Off, (include lpps data).
NA 1           - Append NMEA GPS data On, (Adds GPS to output).
NA 2           - Append NMEA GPS data Off, (no lpps data).
NM 0           - NMEA GPS display, Off, (default), GPS port speed 38400, locked.
NM 1           - NMEA GPS display (RMC + GGA + GSV) data.

```

(continues on next page)



(continued from previous page)

```

NM 2      - NMEA GPS display (ALL) data, use with GPS display applications.
Test Pulser
TE m      - Enable run mode, 0=Off, 1=One cycle, 2=Continuous.
TD m      - Load sample trigger data list, 0=Reset, 1=Singles, 2=Majority.
TV m      - Voltage level at pulse DAC, 0-4095mV, TV=read.
Serial #
SN p n    - Store serial # to flash, p=password, n=(0-65535 BCD).
SN        - Display serial number (BCD).
Status
ST        - Send status line now. This resets the minute timer.
ST 0      - Status line, disabled.
ST 1 m    - Send status line every (m) minutes.(m=1-30, def=5).
ST 2 m    - Include scalar data line, chs S0-S4 after each status line.
ST 3 m    - Include scalar data line, plus reset counters on each timeout.
TI n      - Timer (day hr:min:sec.msec), TI=display time, (TI n=0 clear).
U1 n      - Display Uart error counter, (U1 n=0 to zero counters).
VM 1      - View mode, 0x80=Event_Demarcation_Bit outputs a blank line.
- View mode returns to normal after 'CD','CE','ST' or 'RE'.

H1
Quarknet Scintillator Card, Qnet2.5 Vers 1.11 Compiled Jul 15 2009 HE=Help
Serial#=6531      uC_Volts=3.33      GPS_TempC=0.0      mBar=1023.8

CE        - TMC Counter Enable.
CD        - TMC Counter Disable.
DC        - Display Control Registers, (C0-C3).
WC a d    - Write Control Registers, addr(0-6) data byte(H).
DT        - Display TMC Reg, 0-3, (1=PipeLineDelayRd, 2=PipeLineDelayWr).
WT a d    - Write TMC Reg, addr(1,2) data byte(H), if a=4 write delay word.
DG        - Display GPS Info, Date, Time, Position and Status.
DS        - Display Scalar, channel(S0-S3), trigger(S4), time(S5).
RE        - Reset complete board to power up defaults.
RB        - Reset only the TMC and Counters.
SB p d    - Set Baud,password, 1=19K, 2=38K, 3=57K, 4=115K, 5=230K, 6=460K, 7=920K
SA n      - Save setup, 0=(TMC disable), 1=(TMC enable), 2=(Restore Defaults).
TH        - Thermometer data display (@ GPS), -40 to 99 degrees C.
TL c d    - Threshold Level, signal ch(0-3) (4=setAll), data(0-4095mV), TL=read.
Veto      - Veto select, Off='VE 0', On='VE 1', Gate='VG c', 0-255(D) 10ns/cnt.
View      - View setup registers. Setup=V1, Voltages(V2), GPS LOCK(V3).
HELP      - HE,H1=Page1, H2=Page2, HB=Barometer, HS=Status, HT=Trigger.

VE2
V2
Barometer Pressure Sensor
Calibration Voltage = 1495 mVolts      Use Cmd 'BA' to calibrate.
Sensor Output Voltage= 1655 mVolts      (2.93mV * 565 Cnts)
Pressure mBar       = 1023.6             (1655.5 - 1500)/15 + 1013.25
Pressure inch       = 30.63              (mBar / 33.42)

Timer Capture/Compare Channel
TempC = 0.0      Error? Check sensor cable connection at GPS unit.
TempF = 32.0      (TempC * 1.8) + 32

Analog to Digital Converter Channels(ADC)
Vcc 1.80V = 1.82 vlts      (2.93mV * 621 Cnts)

```

(continues on next page)

(continued from previous page)

```
Vcc 1.20V = 1.19 vlts      (2.93mV * 407 Cnts)
Pos 2.50V = 2.45 vlts      (2.93mV * 837 Cnts)
Neg 5.00V = 5.03 vlts      (7.38mV * 682 Cnts)
Vcc 3.30V = 3.33 vlts      (4.84mV * 689 Cnts)
Pos 5.00V = 4.84 vlts      (7.38mV * 656 Cnts)
5V Test      Max=4.86v      Min=4.84v      Noise=0.015v
```

V3

10 Second Accumulation of 1PPS Latched 25MHz Counter. (20 line buffer)

Buffer	Now (hex)	Prev-Now (dec) (25e6*10)
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	0	0
17	0	0
18	0	0
19	0	0
20	0	0

## 6.1 Subpackages

### 6.1.1 rewrite.lib package

#### 6.1.1.1 Subpackages

#### rewrite.lib.analyzers package

#### Submodules

#### rewrite.lib.analyzers.RateAnalyzer module

```
class rewrite.lib.analyzers.RateAnalyzer.RateAnalyzer (logger=None, head-  

less=True)  

    Bases: object  

    Class that manages the measurement of muon rate.  

    fileWriter()  

    measure_rates (timewindow=5.0, meastime=None)  

        Measure rates seen by the counters. :param timewindow: Time between successive rate measurements in  

        seconds. Default is 5 seconds. :param meastime: Total measurement time in minutes. Default is None.  

    runDaemon()  

    write_rates_to_file (firstline=False)  

        Saves data to file during rate measurements.
```

#### Module contents

## rewrite.lib.common package

### Submodules

#### rewrite.lib.common.CountRecord module

**class** rewrite.lib.common.CountRecord.**CountRecord**(*msg*)

Bases: object

Holds the counting information

incoming format: DS S0=00000000 S1=00000000 S2=00000000 S3=00000000 S4=00000000 S5=18531FFD

##### Parameters

- **(Bool)** (*valid*) – validity of the record. Will be set to True if the message starts with ‘DS’
- **(int)** (*counts\_trigger*) – Counts in channel X
- **(int)** – trigger counts recieved
- **(Real)** (*counts\_time*) – the time of the record

#### rewrite.lib.common.DataRecord module

**class** rewrite.lib.common.DataRecord.**DataRecord**(*msg*)

Bases: object

Record to hold a DataRecords from the DAQ card. Basically just a string wrapper.

**msg** = ''

#### rewrite.lib.common.PressureRecord module

**class** rewrite.lib.common.PressureRecord.**PressureRecord**(*msg*)

Bases: object

Holds Pressure information

incoming format: ‘BA 1495’ or: “mBar now reads = 1015.0 (use cmd ‘SA’ when done)”

##### Parameters

- **(Bool)** (*valid*) – Validity of the record. Set to True, if the message starts with ‘BA’
- **(Real)** (*pressure*) – Floating point value of in the pressure record
- **(PressureType)** (*pressure\_type*) – Either mBar or plain data

**class** rewrite.lib.common.PressureRecord.**PressureType**

Bases: enum.Enum

Type of measured pressure. Plaindata or mBar

**MBAR** = 1

**PLAIN** = 0

## rewrite.lib.common.Record module

**class** `rewrite.lib.common.Record.Record` (*packageName, RecType, timestamp, payload*)

Bases: `object`

The basic data-structure used to communicate between DAQ and analysis

VERY EARLY STAGE! Everything is subject to change!

### Parameters

- **packageName** – A sequential number of all packages send by a DAQ server
- **RecType** – Type of the record
- **timestamp** – Unixtimestamp
- **payload** – Payload to be send

**class** `rewrite.lib.common.Record.RecordType`

Bases: `enum.IntEnum`

Enum of the possible types of Records between DAQ and analysis. Enum for “type-safety”

**CONTROL** = 0

**COUNTER** = 4

**DATA** = 1

**GPS** = 5

**PRESSURE** = 3

**TEMPERATURE** = 2

## rewrite.lib.common.TemperatureRecord module

**class** `rewrite.lib.common.TemperatureRecord.TemperatureRecord` (*msg*)

Bases: `object`

Holds Temperature information

incoming format: TH TH=22.2

### Parameters

- **(Bool)** (*valid*) – Validity of the record. Set to True, if the message starts with ‘TH’
- **(Real)** (*temperature*) – The temperature of the record.

## Module contents

### rewrite.lib.daq package

### Submodules

### rewrite.lib.daq.Connection module

**class** `rewrite.lib.daq.Connection.DAQConnection` (*in\_queue, out\_queue, logger=None*)

Bases: `object`

DAQ Connection class.

Raises `SystemError` if serial connection cannot be established.

**Parameters**

- **logger** (*logging.Logger*) – logger object
- **in\_queue** – input queue
- **out\_queue** – output queue

**Raises** `SystemError`

**get\_serial\_port** ()

Check out which device (/dev/tty) is used for DAQ communication.

Raises `OSError` if binary 'which\_tty\_daq' cannot be found.

**Returns** `serial.Serial` – serial connection port

**Raises** `OSError`

**read** ()

Gets Data from the DAQ card. Read it from the provided queue.

**Returns** `None`

**write** ()

Writes messages from the in queue to the DAQ card

**Returns** `None`

## **rewrite.lib.daq.DAQServer module**

**class** `rewrite.lib.daq.DAQServer.DAQServer`

Bases: `object`

**check\_pressure\_msg** (*msg*)

Check message for pressure information.

**clear\_queues** ()

Clear all the queues filled in `process_incoming()`.

**do** (*msg*)

Send a command to the DAQ card and remove repeated responses from the outqueue if data taking is turned off. Otherwise just send the command to the card.

**get\_gps\_info** ()

**get\_scalars** (*msg=None*)

If `running=True`, read out scalars from the counterqueue. Otherwise, read scalars from given message. Returns the scalar values.

**get\_temp\_and\_pressure** ()

Read out temperature and pressure data. Pressure data in unit counts and mBar. If no measurement is running returns temperature, pressure, pressure\_mbar

**measure\_pulses** (*meastime=None*)

Measure pulses (rising and falling edge times) of trigger events. Using `PulseExtractor` from `muonic`. :param meastime: Total measurement time in minutes. Default is `None`.

**process\_incoming** ()

Sort messages received from the DAQ card and store them in separate queues.

```

read_scalars ()
    Read the scalars of all channels. If no measurement is running, returns scalar values: ch0, ch1, ch2, ch3,
    trigger

reset_scalars ()
    Reset the scalars of all channels.

run ()

setRunning (isRunning)

set_threshold (th_0=300, th_1=300, th_2=300, th_3=300)
    Set the thresholds for the channels of the DAQ card. Default value for all channels is 300.

setup_channel (ch0=False, ch1=False, ch2=False, ch3=False, coincidence='single')
    Enable/Disable channels of the DAQ card and set coincidence settings.

start_reading_data ()
    Start receiving data from the DAQ card and storing it in self.dataqueue.

stop ()

stop_reading_data ()
    Stop receiving data from the DAQ card.

```

## rewrite.lib.daq.Exceptions module

Utility classes and functions needed by DAQ related modules

```

exception rewrite.lib.daq.Exceptions.DAQIOError
    Bases: OSError
    DAQ IOError Exception class

exception rewrite.lib.daq.Exceptions.DAQMissingDependencyError
    Bases: Exception
    Exception class which is thrown if runtime dependencies are not met

```

## rewrite.lib.daq.Provider module

```

class rewrite.lib.daq.Provider.DAQProvider (logger=None)
    Bases: object
    Class providing the public API and helpers for the communication with the DAQ card

    LINE_PATTERN = re.compile ("^[a-zA-Z0-9+-. , : () = $ / # ? ! % _ @ * | ~ ' ] * [ \n \r ] * $")

    data_available ()
        Tests if data is available from the DAQ card.

        Returns int or bool

    get (*args)
        Get data from the DAQ card.

        Raises DAQIOError if the queue is empty.

        Parameters args (list) – queue arguments

        Returns str or None – next item from the queue

```

**Raises** DAQIOError

**put** (*\*args*)

Senf data to the DAQ card.

**Parameters** **args** (*list*) – queue arguments

**Returns** None

**validate\_line** (*line*)

Validate line against regex pattern. Returns None if the provided line is invalid or the line if it is valid.

**Parameters** **line** (*str*) – line to validate

**Returns** str or None

## rewrite.lib.daq.getDevice module

rewrite.lib.daq.getDevice.**get\_Device**()

Reads dmesg and searches for the daq card. Then returns its name.

returns ttyUSB0 by default

**Returns** name of the device file in /dev/ or ttyUSB0 by default

## Module contents

### rewrite.lib.utils package

### Subpackages

### rewrite.lib.utils.db package

### Submodules

## rewrite.lib.utils.db.CountRecordAdapter module

**class** rewrite.lib.utils.db.CountRecordAdapter.**CountRecordAdapter** (*\*args*,  
\*\**kwargs*)

Bases: mongoengine.document.EmbeddedDocument

Adapter class to store Counts in MongoDB

### Parameters

- **(Bool)** (*valid*) – validity of the record.
- **(int)** (*counts\_trigger*) – Counts in channel X
- **(int)** – trigger counts recieved
- **(Real)** (*counts\_time*) – the time of the record

**counters\_time**

Counts in the time register of the DAQ card. Basically a timestamp

**counts\_ch0**

Counts in channel 0



**counts\_ch1**  
Counts in channel 1

**counts\_ch2**  
Counts in channel 2

**counts\_ch3**  
Counts in channel 3

**counts\_trigger**  
Trigger counts

**createCount ()**  
Creates a CountRecord from the current object  
**Returns** CountRecord from the current CountRecordAdapter

**static get (rec)**  
Creates a CountRecordAdapter from a CountRecord  
**Parameters** **rec** – CountRecord to convert

**valid**  
Set to true if the underlying record is valid

### rewrite.lib.utils.db.DataRecordAdapter module

**class** `rewrite.lib.utils.db.DataRecordAdapter.DataRecordAdapter` (\*args, \*\*kwargs)  
Bases: `mongoengine.document.EmbeddedDocument`  
Adapter class for the data record. Basically just a string wrapper.

**createData ()**  
Converts the current object to a DataRecord  
**Returns** DataRecord from the current DataRecordAdapter

**static get (rec)**  
Create a DataRecordAdapter from a DataRecord  
**Parameters** **rec** – DataRecord to convert

**msg**  
A unicode string field.

### rewrite.lib.utils.db.PressureRecordAdapter module

**class** `rewrite.lib.utils.db.PressureRecordAdapter.PressureRecordAdapter` (\*args, \*\*kwargs)  
Bases: `mongoengine.document.EmbeddedDocument`  
Adapter class for the pressure record.

**Parameters**

- **(Bool)** (*valid*) – Validity of the record. Set to True, if the message starts with ‘BA’
- **(Real)** (*pressure*) – Floating point value of in the pressure record
- **(PressureType)** (*pressure\_type*) – Either mBar or plain data

**createPressure ()**

Converts the current object to a PressureRecord

**Returns** PressureRecord from the current PressureRecordAdapter

**static get (rec)**

Create a PressureRecordAdapter from a PressureRecord

**Parameters** **rec** – PressureRecord to convert

**pressure**

Fixed-point decimal number field. Stores the value as a float by default unless *force\_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

**pressure\_type**

A unicode string field.

**valid**

Boolean field type.

**rewrite.lib.utils.db.RecordAdapter module**

```
class rewrite.lib.utils.db.RecordAdapter.RecordAdapter (**kwargs)
```

Bases: `mongoengine.document.Document`

This is an adapter class which helps to save and load a record in MongoDB.

**Parameters**

- **\_id** – Object ID given by MongoDB. Explicitly declared to be able to load from dict.
- **packageName** – A sequential number of all packages send by a DAQ server
- **RecType** – Type of the record
- **timestamp** – Unixtimestamp
- **payload\_cnt** – Count Payload
- **payload\_dat** – Data Payload
- **payload\_tme** – Temperature Payload
- **payload\_prs** – Pressure Payload

Sadly the payload for each type of payload needs to be in a separate field, as we need an EmbeddedDocument-Field of a certain type.

**exception DoesNotExist**

Bases: `mongoengine.errors.DoesNotExist`

**exception MultipleObjectsReturned**

Bases: `mongoengine.errors.MultipleObjectsReturned`

**createRecord ()**

Creates a Record with the current data.

**Returns** Record with the current data

**static get (rec)**

Construct a RecordAdapter from a Record

**Parameters** **rec** – a record that will be converted to a RecordAdapter

**Returns** a newly constructed RecordAdapter

**static** `getChoice(i)`

Translate RecordType aka int to string.

**Parameters** `i` – RecordType to be converted.

**id**

A field wrapper around MongoDB's ObjectIds.

**objects**

The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet`, probably the same one that was passed in, but modified in some way.

**packageNumber**

32-bit integer field.

**payload\_cnt**

An embedded document field - with a declared `document_type`. Only valid values are subclasses of `EmbeddedDocument`.

**payload\_dat**

An embedded document field - with a declared `document_type`. Only valid values are subclasses of `EmbeddedDocument`.

**payload\_prs**

An embedded document field - with a declared `document_type`. Only valid values are subclasses of `EmbeddedDocument`.

**payload\_tem**

An embedded document field - with a declared `document_type`. Only valid values are subclasses of `EmbeddedDocument`.

**timestamp**

Datetime field.

Uses the python-dateutil library if available alternatively use `time.strptime` to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: To default the field to the current datetime, use: `DateTimeField(default=datetime.utcnow)`

**Note: Microseconds are rounded to the nearest millisecond.** Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

**type**

only valid choices here are: CONTROL, DATA, TEMPERATURE, PRESSURE, COUNTER

**Type** String

## rewrite.lib.utils.db.TemperatureRecordAdapter module

**class** `rewrite.lib.utils.db.TemperatureRecordAdapter.TemperatureRecordAdapter(*args, **kwargs)`

Bases: `mongoengine.document.EmbeddedDocument`

Adapter class for the temperature record.

**Parameters**

- **(Bool)** (*valid*) – Validity of the record. Set to True, if the message starts with ‘TH’
- **(Real)** (*temperature*) – The temperature of the record.

**createTemperature()**

Converts the current object to a TemperatureRecord

**Returns** TemperatureRecord from the current TemperatureRecordAdapter

**static get(rec)**

Create a TemperatureRecordAdapter from a TemperatureRecord

**Parameters** **rec** – TemperatureRecord to convert

**temperature**

Fixed-point decimal number field. Stores the value as a float by default unless *force\_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

**valid**

Boolean field type.

**Module contents****Submodules****rewrite.lib.utils.WriterToMongoDB module**

**class** rewrite.lib.utils.WriterToMongoDB.**WriterToMongoDB** (*logger=None*)

Bases: object

Writes incoming data to the MongoDB for storage

**DBWriter()****runDaemon()****rewrite.lib.utils.ReaderFromMongoDB module**

**class** rewrite.lib.utils.ReaderFromMongoDB.**ReaderFromMongoDB** (*logger=None*)

Bases: object

Class that read data from MongoDB and sends it as if it were coming from a DAQ card. This is a basic version, which can be extended. Inits the MongoDB connection and the zeromq socket. This needs to start before any analysis.

**clear\_queues()**

Fake function. Just for API compatibility

**do(arg)**

Fake function. Just for API compatibility

**get\_temp\_and\_pressure()**

Fake function. Just for API compatibility

**read\_scalars()**

Fake function. Just for API compatibility

```

reset_scalars ()
    Fake function. Just for API compatibility

run ()
    Get data from a certain timeframe from the db and then sends it through the socket.

setRunning (state)
    Fake function. Just for API compatibility

set_threshold (ch0, ch1, ch2, ch3)
    Fake function. Just for API compatibility

setup_channel (ch0, ch1, ch2, ch3, coincidence)
    Fake function. Just for API compatibility

start_reading_data ()
    Fake function. Just for API compatibility

stop_reading_data ()
    Fake function. Just for API compatibility

```

## Module contents

### 6.1.1.2 Submodules

#### 6.1.1.3 `rewrite.lib.Skyview` module

```

class rewrite.lib.Skyview.Skyview
    Bases: object

    calculate_rates ()
        Calculate rates during rate measurements.

    check_pressure_msg (msg)
        Check message for pressure information.

    clear_queues ()
        Clear all the queues filled in process_incoming().

    do (msg)
        Send command to DAQ card and remove repeated response from the outqueue if data taking is turned off.
        Otherwise just send command to DAQ card.

    get_gps_info ()

    get_scalars (msg=None)
        If running=True, read out scalars from the counterqueue. Otherwise, read scalars from given message.
        Returns the scalar values.

    get_temp_and_pressure ()
        Read out temperature and pressure data. Pressure data in unit counts and mBar. If no measurement is
        running returns temperature, pressure, pressure_mbar

    measure_pulses (meastime=None)
        Measure pulses (rising and falling edge times) of trigger events. Using PulseExtractor from muonic.
        :param meastime: Total measurement time in minutes. Default is None.

    measure_rates (timewindow=5.0, meastime=None)
        Measure rates seen by the counters. :param timewindow: Time between successive rate measurements in
        seconds. Default is 5 seconds. :param meastime: Total measurement time in minutes. Default is None.

```

**process\_incoming()**  
Sort messages received from the DAQ card and store them in separate queues.

**read\_scalars()**  
Read the scalars of all channels. If no measurement is running, returns scalar values: ch0, ch1, ch2, ch3, trigger

**reset\_scalars()**  
Reset the scalars of all channels.

**set\_threshold(th\_0=300, th\_1=300, th\_2=300, th\_3=300)**  
Set the thresholds for the channels of the DAQ card. Default value for all channels is 300.

**setup\_channel(ch0=False, ch1=False, ch2=False, ch3=False, coincidence='single')**  
Enable/Disable channels of the DAQ card and set coincidence settings.

**start\_reading\_data()**  
Start receiving data from the DAQ card and storing it in self.dataqueue.

**stop\_reading\_data()**  
Stop receiving data from the DAQ card.

**write\_rates\_to\_file(filename="", firstline=False)**  
Saves data to file during rate measurements.

#### 6.1.1.4 Module contents

## 6.2 Submodules

## 6.3 rewrite.example\_measurement module

## 6.4 rewrite.runRates module

**rewrite.runRates.run()**  
Creates an instance of RateAnalyzer and runs a simple rate measurement.

## 6.5 rewrite.runServer module

**class** **rewrite.runServer.RequestHandler**(request, client\_address, server)  
Bases: xmlrpc.server.SimpleXMLRPCRequestHandler  
Adapter Class for xmlrpc

**rpc\_paths = ('/RPC2',)**

**rewrite.runServer.run()**  
Starts an instance of the DAQ server with xmlrpc enabled and then enters an infinite loop and processes requests

## 6.6 rewrite.runWriterToMongoDB module

**rewrite.runWriterToMongoDB.run()**

## 6.7 rewrite.simpleClient module

```
rewrite.simpleClient.reciever_loop()  
rewrite.simpleClient.run()
```

## 6.8 Module contents





## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### r

- [rewrite](#), [27](#)
- [rewrite.lib](#), [26](#)
- [rewrite.lib.analyzers](#), [15](#)
- [rewrite.lib.analyzers.RateAnalyzer](#), [15](#)
- [rewrite.lib.common](#), [17](#)
- [rewrite.lib.common.CountRecord](#), [16](#)
- [rewrite.lib.common.DataRecord](#), [16](#)
- [rewrite.lib.common.PressureRecord](#), [16](#)
- [rewrite.lib.common.Record](#), [17](#)
- [rewrite.lib.common.TemperatureRecord](#),  
[17](#)
- [rewrite.lib.daq](#), [20](#)
- [rewrite.lib.daq.Connection](#), [17](#)
- [rewrite.lib.daq.DAQServer](#), [18](#)
- [rewrite.lib.daq.Exceptions](#), [19](#)
- [rewrite.lib.daq.getDevice](#), [20](#)
- [rewrite.lib.daq.Provider](#), [19](#)
- [rewrite.lib.Skyview](#), [25](#)
- [rewrite.lib.utils](#), [25](#)
- [rewrite.lib.utils.db](#), [24](#)
- [rewrite.lib.utils.db.CountRecordAdapter](#),  
[20](#)
- [rewrite.lib.utils.db.DataRecordAdapter](#),  
[21](#)
- [rewrite.lib.utils.db.PressureRecordAdapter](#),  
[21](#)
- [rewrite.lib.utils.db.RecordAdapter](#), [22](#)
- [rewrite.lib.utils.db.TemperatureRecordAdapter](#),  
[23](#)
- [rewrite.lib.utils.ReaderFromMongoDB](#), [24](#)
- [rewrite.lib.utils.WriterToMongoDB](#), [24](#)
- [rewrite.runRates](#), [26](#)
- [rewrite.runServer](#), [26](#)
- [rewrite.runWriterToMongoDB](#), [26](#)
- [rewrite.simpleClient](#), [27](#)



## Symbols

-d  
muonic command line option, 7

-n  
muonic command line option, 7

-p  
muonic command line option, 7

-s  
muonic command line option, 7

-t sec  
muonic command line option, 7

## A

automatically write a file with  
pulsetimes in a non hexadecimal  
representation  
muonic command line option, 7

## C

calculate\_rates() (rewrite.lib.Skyview.Skyview  
method), 25

change the timewindow for the  
calculation of the rates. If  
you expect very low rates, you  
might consider to change it to  
larger values.  
muonic command line option, 7

check\_pressure\_msg()  
(rewrite.lib.daq.DAQServer.DAQServer  
method), 18

check\_pressure\_msg()  
(rewrite.lib.Skyview.Skyview method), 25

clear\_queues() (rewrite.lib.daq.DAQServer.DAQServer  
method), 18

clear\_queues() (rewrite.lib.Skyview.Skyview  
method), 25

clear\_queues() (rewrite.lib.utils.ReaderFromMongoDB.  
method), 24

CONTROL (rewrite.lib.common.Record.RecordType at-  
tribute), 17

COUNTER (rewrite.lib.common.Record.RecordType at-  
tribute), 17

counters\_time(rewrite.lib.utils.db.CountRecordAdapter.CountRecordA  
tribute), 20

CountRecord (class in  
rewrite.lib.common.CountRecord), 16

CountRecordAdapter (class in  
rewrite.lib.utils.db.CountRecordAdapter),  
20

counts\_ch0(rewrite.lib.utils.db.CountRecordAdapter.CountRecordAdap  
tribute), 20

counts\_ch1(rewrite.lib.utils.db.CountRecordAdapter.CountRecordAdap  
tribute), 20

counts\_ch2(rewrite.lib.utils.db.CountRecordAdapter.CountRecordAdap  
tribute), 21

counts\_ch3(rewrite.lib.utils.db.CountRecordAdapter.CountRecordAdap  
tribute), 21

counts\_trigger(rewrite.lib.utils.db.CountRecordAdapter.CountRecordA  
tribute), 21

createCount() (rewrite.lib.utils.db.CountRecordAdapter.CountRecordA  
method), 21

createData() (rewrite.lib.utils.db.DataRecordAdapter.DataRecordAdap  
method), 21

createPressure() (rewrite.lib.utils.db.PressureRecordAdapter.Pressur  
method), 21

createRecord() (rewrite.lib.utils.db.RecordAdapter.RecordAdapter  
method), 22

createTemperature()  
(rewrite.lib.utils.db.TemperatureRecordAdapter.TemperatureReco  
method), 24

## D

DAQConnection (class in rewrite.lib.daq.Connection),  
17

DAQIOError, 19

DAQMissingDependencyError, 19

DBReaderFromMongoDB (class in rewrite.lib.daq.Provider), 19

DAQServer (class in rewrite.lib.daq.DAQServer), 18

DATA (rewrite.lib.common.Record.RecordType attribute),  
17

`data_available()` (*rewrite.lib.daq.Provider.DAQProvider* method), 19  
`data_available_and_pressure()` (*rewrite.lib.Skyview.Skyview* method), 25  
`DataRecord` (class *rewrite.lib.common.DataRecord*), 16  
`in get_temp_and_pressure()` (*rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB* method), 24  
`DataRecordAdapter` (class *rewrite.lib.utils.db.DataRecordAdapter*), 21  
`in getChoice()` (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* static method), 22  
`DBWriter()` (*rewrite.lib.utils.WriterToMongoDB.WriterToMongoDB* method), 24  
`in write()` (*rewrite.lib.common.Record.RecordType* attribute), 17  
`debug mode.` Use it to generate more log messages on the console.  
    muonic command line option, 7  
`default is 5 seconds.`  
    muonic command line option, 7  
`do()` (*rewrite.lib.daq.DAQServer.DAQServer* method), 18  
`do()` (*rewrite.lib.Skyview.Skyview* method), 25  
`do()` (*rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB* method), 24  
**F**  
`fileWriter()` (*rewrite.lib.analyzers.RateAnalyzer.RateAnalyzer* method), 15  
**G**  
`get()` (*rewrite.lib.daq.Provider.DAQProvider* method), 19  
`get()` (*rewrite.lib.utils.db.CountRecordAdapter.CountRecordAdapter* static method), 21  
`get()` (*rewrite.lib.utils.db.DataRecordAdapter.DataRecordAdapter* static method), 21  
`get()` (*rewrite.lib.utils.db.PressureRecordAdapter.PressureRecordAdapter* static method), 22  
`get()` (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* static method), 22  
`get()` (*rewrite.lib.utils.db.TemperatureRecordAdapter.TemperatureRecordAdapter* static method), 24  
`get_Device()` (in module *rewrite.lib.daq.getDevice*), 20  
`get_gps_info()` (*rewrite.lib.daq.DAQServer.DAQServer* method), 18  
`get_gps_info()` (*rewrite.lib.Skyview.Skyview* method), 25  
`get_scalars()` (*rewrite.lib.daq.DAQServer.DAQServer* method), 18  
`get_scalars()` (*rewrite.lib.Skyview.Skyview* method), 25  
`get_serial_port()` (*rewrite.lib.daq.Connection.DAQConnection* method), 18  
`get_temp_and_pressure()` (*rewrite.lib.daq.DAQServer.DAQServer* method), 18  
`get_temp_and_pressure()` (*rewrite.lib.Skyview.Skyview* method), 25  
**L**  
`LINE_PATTERN` (*rewrite.lib.daq.Provider.DAQProvider* attribute), 19  
**M**  
`MBAR` (*rewrite.lib.common.PressureRecord.PressureType* attribute), 16  
`measure_pulses()` (*rewrite.lib.daq.DAQServer.DAQServer* method), 18  
`measure_pulses()` (*rewrite.lib.Skyview.Skyview* method), 25  
`measure_rates()` (*rewrite.lib.analyzers.RateAnalyzer.RateAnalyzer* method), 15  
`measure_rates()` (*rewrite.lib.Skyview.Skyview* method), 25  
`read()` (*rewrite.lib.common.DataRecord.DataRecord* attribute), 16  
`read()` (*rewrite.lib.utils.db.DataRecordAdapter.DataRecordAdapter* attribute), 21  
`muonic command line option`  
    -d, 7  
    -p, 7  
    -s, 7  
    -t sec, 7  
`automatically write a file with`  
    pulsetimes in a non hexadecimal representation, 7  
`change the timewindow for the`  
    calculation of the rates. If you expect very low rates, you might consider to change it to larger values., 7  
`debug mode.` Use it to generate more log messages on the console., 7  
`default is 5 seconds., 7`  
`supress any status messages in the`  
    output raw data file, might be useful if you want use muonic

- only for data taking and use another script afterwards for analysis., 7
- use the simulation mode of muonic (no real data, so no physics behind!). This should only be used for testing and developing the software, 7
- ## O
- objects (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* attribute), 23
- ## P
- packageName (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* attribute), 23
- payload\_cnt (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* attribute), 23
- payload\_dat (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* attribute), 23
- payload\_prs (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* attribute), 23
- payload\_tem (*rewrite.lib.utils.db.RecordAdapter.RecordAdapter* attribute), 23
- PLAIN (*rewrite.lib.common.PressureRecord.PressureType* attribute), 16
- PRESSURE (*rewrite.lib.common.Record.RecordType* attribute), 17
- pressure (*rewrite.lib.utils.db.PressureRecordAdapter.PressureRecordAdapter* attribute), 22
- pressure\_type (*rewrite.lib.utils.db.PressureRecordAdapter.PressureRecordAdapter* attribute), 22
- PressureRecord (class in *rewrite.lib.common.PressureRecord*), 16
- PressureRecordAdapter (class in *rewrite.lib.utils.db.PressureRecordAdapter*), 21
- PressureType (class in *rewrite.lib.common.PressureRecord*), 16
- process\_incoming() (*rewrite.lib.daq.DAQServer.DAQServer* method), 18
- process\_incoming() (*rewrite.lib.Skyview.Skyview* method), 25
- put() (*rewrite.lib.daq.Provider.DAQProvider* method), 20
- read\_scalars() (*rewrite.lib.Skyview.Skyview* method), 26
- read\_scalars() (*rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB* method), 24
- ReaderFromMongoDB (class in *rewrite.lib.utils.ReaderFromMongoDB*), 24
- receiver\_loop() (in module *rewrite.simpleClient*), 27
- Record (class in *rewrite.lib.common.Record*), 17
- RecordAdapter (class in *rewrite.lib.utils.db.RecordAdapter*), 22
- RecordAdapter.DoesNotExist, 22
- RecordAdapter.MultipleObjectsReturned, 22
- RecordType (class in *rewrite.lib.common.Record*), 17
- RequestHandler (class in *rewrite.runServer*), 26
- reset\_scalars() (*rewrite.lib.daq.DAQServer.DAQServer* method), 19
- reset\_scalars() (*rewrite.lib.Skyview.Skyview* method), 26
- reset\_scalars() (*rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB* method), 24
- rewrite (module), 27
- rewrite.lib (module), 26
- rewrite.lib.analyzers (module), 15
- rewrite.lib.analyzers.RateAnalyzer (module), 15
- rewrite.lib.common (module), 17
- rewrite.lib.common.CountRecord (module), 16
- rewrite.lib.common.DataRecord (module), 16
- rewrite.lib.common.PressureRecord (module), 16
- rewrite.lib.common.Record (module), 17
- rewrite.lib.common.TemperatureRecord (module), 17
- rewrite.lib.daq (module), 20
- rewrite.lib.daq.Connection (module), 17
- rewrite.lib.daq.DAQServer (module), 18
- rewrite.lib.daq.Exceptions (module), 19
- rewrite.lib.daq.getDevice (module), 20
- rewrite.lib.daq.Provider (module), 19
- rewrite.lib.Skyview (module), 25
- rewrite.lib.utils (module), 25
- rewrite.lib.utils.db (module), 24
- rewrite.lib.utils.db.CountRecordAdapter (module), 20
- rewrite.lib.utils.db.DataRecordAdapter (module), 21
- rewrite.lib.utils.db.PressureRecordAdapter (module), 21
- rewrite.lib.utils.db.RecordAdapter (module), 22
- rewrite.lib.utils.db.TemperatureRecordAdapter

(module), 23  
 rewrite.lib.utils.ReaderFromMongoDB (module), 24  
 rewrite.lib.utils.WriterToMongoDB (module), 24  
 rewrite.runRates (module), 26  
 rewrite.runServer (module), 26  
 rewrite.runWriterToMongoDB (module), 26  
 rewrite.simpleClient (module), 27  
 rpc\_paths (rewrite.runServer.RequestHandler attribute), 26  
 run() (in module rewrite.runRates), 26  
 run() (in module rewrite.runServer), 26  
 run() (in module rewrite.runWriterToMongoDB), 26  
 run() (in module rewrite.simpleClient), 27  
 run() (rewrite.lib.daq.DAQServer.DAQServer method), 19  
 run() (rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB method), 25  
 runDaemon() (rewrite.lib.analyzers.RateAnalyzer.RateAnalyzer method), 15  
 runDaemon() (rewrite.lib.utils.WriterToMongoDB.WriterToMongoDB method), 24

## S

set\_threshold() (rewrite.lib.daq.DAQServer.DAQServer method), 19  
 set\_threshold() (rewrite.lib.Skyview.Skyview method), 26  
 set\_threshold() (rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB method), 25  
 setRunning() (rewrite.lib.daq.DAQServer.DAQServer method), 19  
 setRunning() (rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB method), 25  
 setup\_channel() (rewrite.lib.daq.DAQServer.DAQServer method), 19  
 setup\_channel() (rewrite.lib.Skyview.Skyview method), 26  
 setup\_channel() (rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB method), 25  
 Skyview (class in rewrite.lib.Skyview), 25  
 start\_reading\_data() (rewrite.lib.daq.DAQServer.DAQServer method), 19  
 start\_reading\_data() (rewrite.lib.Skyview.Skyview method), 26  
 start\_reading\_data() (rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB method), 25  
 stop() (rewrite.lib.daq.DAQServer.DAQServer method), 19  
 stop\_reading\_data() (rewrite.lib.daq.DAQServer.DAQServer method), 19  
 stop\_reading\_data() (rewrite.lib.Skyview.Skyview method), 26  
 stop\_reading\_data() (rewrite.lib.utils.ReaderFromMongoDB.ReaderFromMongoDB method), 25  
 stop\_reading\_data() (rewrite.lib.utils.db.TemperatureRecordAdapter.TemperatureRecordAdapter method), 24  
 stop\_reading\_data() (rewrite.lib.utils.db.TemperatureRecordAdapter.TemperatureRecordAdapter method), 25  
 suppress any status messages in the output raw data file, might be useful if you want use muonic only for data taking and use another script afterwards for analysis.  
 muonic command line option, 7

## T

TEMPERATURE (rewrite.lib.common.Record.RecordType attribute), 17  
 Temperature (rewrite.lib.utils.db.TemperatureRecordAdapter.TemperatureRecordAdapter attribute), 24  
 TemperatureRecord (class in rewrite.lib.common.TemperatureRecord), 23  
 TemperatureRecordAdapter (class in rewrite.lib.utils.db.TemperatureRecordAdapter), 23  
 timestamp (rewrite.lib.utils.db.RecordAdapter.RecordAdapter attribute), 23  
 type (rewrite.lib.utils.db.RecordAdapter.RecordAdapter attribute), 23

## U

use the simulation mode of muonic (no real data, so no physics) . This should only used for testing and developing the software  
 muonic command line option, 7

## V

valid (rewrite.lib.utils.db.CountRecordAdapter.CountRecordAdapter attribute), 21  
 valid (rewrite.lib.utils.db.PressureRecordAdapter.PressureRecordAdapter attribute), 22  
 valid (rewrite.lib.utils.db.TemperatureRecordAdapter.TemperatureRecordAdapter attribute), 24  
 validate\_line() (rewrite.lib.daq.Provider.DAQProvider method), 20

## W

write() (rewrite.lib.daq.Connection.DAQConnection method), 18  
 write\_rates\_to\_file() (rewrite.lib.analyzers.RateAnalyzer.RateAnalyzer method), 15



`write_rates_to_file()`  
    (*rewrite.lib.Skyview.Skyview method*), 26  
`WriterToMongoDB` (class *in*  
    *rewrite.lib.utils.WriterToMongoDB*), 24