Miscellaneous programs for Miniball

Nigel Warr

11 July 2010

Contents

1 The program sort_22Ne

In order to determine the positions of the Miniball detectors, we take data with a ^{22}Ne beam on a deuterated polyethylene target and look at the 440 keV peak from the d(^{22}Ne, ^{23}Na)n reaction, using its doppler shifted energy for each segment to determine the angles.

As the segment efficiency is quite low, we rely only on the core energies but use the segment as a gate. We assume that the segment with the most energy is the one where the first interaction occurred.

So we want to sort a Miniball Event Data (MED) file generating spectra for the core gated by the six segments, for each core. i.e. 144 spectra for the 24 cores.

The program *sort_22Ne* does this.

1.1 Usage

The simplest use is:

```
sort_22Ne XXX.med
```

Note that you can specify more than one MED file on the command line.

It uses the default calibration for approximately 4 MeV range (unless you specify a calibration file) and sorts the data creating a root file called *sort_22Ne.root*, which contains the 144 TH1I histograms. These have names like det0_gate1, which means detector zero gated by segment 1 of that detector.

If you wish a different output file, you can specify it with the *-o outputfile.root* option on the command line.

1.1.1 Calibrating

If you wish to specify a calibration file, you can use either the Marabou format or the offl_root_med format and *sort_22Ne* will automatically detect which type it is. To use a calibration file, use the *-c calibration_file* option on the command line.

2 The script sort_crosstalk

In order to check if there is crosstalk on a Miniball capsule, we look at the core signal gated by each of the six segments. Since we are looking at the same

signal, the energy spectrum *should* be identical in each case. However, if there is crosstalk, the energy in the core may depend on which segment is hit. So we need segment-gated core histograms, which is exactly what sort_22Ne gives us. The only difference is that we run it on source data, rather than in-beam data. Also we run it with the -a switch in order to prevent subtraction of the off window.

So the sort_crosstalk script just calls sort_22Ne with the -a switch in order to turn off subtraction of the off window from the on window and with -o sort_crosstalk.root to change the default output file name.
To analyse it, it is just enough to superimpose all the histograms for one detector, which can be done in root or with hdtv.

3 The program sort_addback

In order to determine the absolute efficiency with and without addback, it is necessary to generate per-detector histograms of the core signals and also per-cluster histograms of the sum of the three detectors in a cluster (addback). Generating addback histograms involves correlating the timestamps to match events in the same cluster, but in different DGFs.

The program *sort_addback* does this. It takes one or more Miniball Event Data (MED) file and generates these histograms. The per-detector histograms use the same naming convention as the Marabou root files. i.e. the core spectra are called *hEXXcCal* where XX indicates the cluster number (1... 8) and which DGF (1, 3 or 5) for that cluster. It also has the sum of cores *hCoreSum*. In addition to these, it has *hEaddbackX* for the addback spectrum for cluster X (1... 8) and *hCluSum* for the sum of cores with addback. The use of this naming convention makes it possible to use either files generated by this program or those saved by Marabou for determining the efficiency without addback.

3.1 Usage

The simplest use is:

```
sort_addback XXX.med
```

Note that you can specify more than one MED file on the command line.

By default a file called *sort_addback.root* is created. If you wish a different output file, you can specify it with the *-o outputfile.root* option on the command line.

3.1.1 Calibrating

If you wish to specify a calibration file, you can use either the Marabou format or the offl_root_med format and *sort_addback* will automatically detect which type it is. To use a calibration file, use the *-c calibration_file* option on the command line.

4 The program multiplot

It is often useful to be able to quickly plot the same histogram from several different root files with the same limits in order to compare them. The program *multiplot* is a simple script to do this.

4.1 Usage

The first parameter on the command line is the name of the histogram and all subsequent parameters are root files. It will create a TCanvas and divide it up plotting that histogram from each of the root files.

The *-x xlo:xhi* switch sets the x-limits. e.g. -x100:1000 will plot the x range from 100 to 1000.

The *-y ylo:yhi* switch sets the y-limits. e.g. -y100:1000 will plot the y range from 100 to 1000.

The *-l* switch selects a logarithmic scale for y for 1D histograms or for z for 2D histograms.

The *-r rebin_fact* switch allows you to rebin the histogram before plotting.

If you plot a 2D histogram, you may want to pass options like "colz" to the Draw function. This is done using the *-o colz* option.

Once you've created the TCanvas, you can easily save it as a .jpeg or .pdf etc. You also have control over the objects on the canvas, so you can manipulate them with the mouse. e.g. change limits, switch log on/off etc.

5 The program analyse_22Ne

In order to determine the positions of the Miniball detectors, we take data with a ^{22}Ne beam on a deuterated polyethylene target and look at the 440 keV peak

from the d(^{22}Ne, ^{23}Na)n reaction, using its doppler shifted energy for each segment to determine the angles.

The program *sort_22Ne* is used to sort the data into histograms, but then it is necessary to determine the centroid of the doppler-shifted 440 keV peak for each spectrum. This is done by the program *analyse_22Ne*.

It takes the *sort_22Ne.root* file generated by *sort_22Ne* (you can use the *-i inputfile.root* option to select a different root file). It then fits the doppler-shifted 440 keV peak in each spectrum and generates a TCanvas for each cluster in which it displays the fit for all 18 segments in a 3 x 6 grid. It writes the values it obtains to a file *analyse_22Ne.dat* (the *-o outputfile* switch allows you to change this filename).

Each TCanvas is also saved in .pdf and .png format.

If you are working on a slow connection, you might find it useful to use the *-b* switch which turns on batch mode. Then, the output files are all generated, including the .pdf and .png copies of the TCanvas, but the TCanvas is not actually displayed.

6 The program analyse_60Co

The program *analyse_60Co* is used to determine the efficiency of Miniball using data sorted by *sort_60Co* or the marabou root file. Note, however, that it is only possible to determine the efficiency with addback in the former case.

The method relies on the sum peak of the 1173.2 and 1332.5 keV lines. The area for this peak is proportional to the product of the efficiencies at 1173.2 and 1332.5 keV. So dividing by the area of the peak at 1173.2 keV we get the efficiency at 1332.5 keV.

Strictly, we should fit each spectrum at 1173.2 and 2505.7 keV and calculate the ratio and then sum them. However, this gives problems of statistics with the sum peak. So instead, we assume that the average ratio for each detector is equal to the ratio in the spectrum which is the sum of all detectors. This is not strictly valid, but it is not a bad approximation if the individual detectors have similar efficiencies. The total efficiency is then the number of detectors multiplied by this average efficiency per detector.

The program fits the sum spectra (with and without addback if the addback spectra are available) and creates a TCanvas for each of the three peaks (1173.2, 1332.5 and 2502.7 keV), where it shows the fits for each core in a 6 x 4 grid and an additional TCanvas for the sum of all cores and sum of all clusters (i.e.

without and with addback respectively) for the three peaks in a 2 x 3 grid.

The program uses the file *sort_addback.root* generated by the program *sort_addback* unless you use the *-i inputfile.root* option to specify a different file.

The results are written to *analyse_60Co.da* (the *-o outputfile* switch allows you to change this filename).

Each TCanvas is also saved in .pdf and .png format.

If you are working on a slow connection, you might find it useful to use the *-b* switch which turns on batch mode. Then, the output files are all generated, including the .pdf and .png copies of the TCanvas, but the TCanvas is not actually displayed.

7 The program analyse_152Eu

In order to determine the relative efficiency of Miniball, we take data with a ^{152}Eu source at the target position and sort it using *sort_addback*. Note that it is also possible to use the root files created by marabou, but then there is no addback data.

We then analyse this data using *analyse_152Eu*. By default it uses the *sort_addback.root* file which is the default name for the output file of *sort_addback*. The *-i inputfile* switch can be used to specify a diffeent one. By default the results are written to *analyse_152Eu.dat*, but this can be changed using the *-o outputfile* option.

The program opens a TCanvas divided up into a grid showing the fits of the main peaks of the spectrum without addback and a second one with addback and then a third with the efficiency as a function of energy.

If you know the absolute efficiency without addback at 1332 keV, you can specify this using the *-n normalisation* switch. In this case, *analyse_152Eu* displays the absolute efficiencies rather than the relative ones.

If you are working on a slow connection, you might find it useful to use the *-b* switch which turns on batch mode. Then, the output files are all generated, including the .pdf and .png copies of the TCanvas, but the TCanvas is not actually displayed.

8 The program analyse_EuBa

In order to determine the relative efficiency of Miniball, we take data with ^{133}Ba and ^{152}Eu sources at the target position simultaneously (back to back) and sort it using *sort_addback*. Note that it is also possible to use the root files created by marabou, but then there is no addback data.

We then analyse this data using *analyse_EuBa*. By default it uses the *sort_addback.root* file which is the default name for the output file of *sort_addback*. The *-i inputfile* switch can be used to specify a diffeent one. By default the results are written to *analyse_EuBa.dat*, but this can be changed using the *-o outputfile* option.

The program opens a TCanvas divided up into a grid showing the fits of the main peaks of the spectrum without addback and a second one with addback and then a third with the efficiency as a function of energy.

If you know the absolute efficiency without addback at 1332 keV, you can specify this using the *-n normalisation* switch. In this case, *analyse_EuBa* displays the absolute efficiencies rather than the relative ones.

In order to match the ^{152}Eu peaks to the ^{133}Ba peaks, the code needs to know the absolute activities of the two sources. As it does not have access to the live time for the measurement, it cannot use this data for an absolute normalisation, but assumes that as the data were acquired simultaneously, they have the same live time and uses the ratio to calculate the relative factor between the two sources.

By default, the absolute activities of the two ISOLDE sources 3680RP (^{133}Ba) and 3687RP (^{152}Eu) are assumed.

The numbers were taken from the ISOLDE source list information:

```
3680RP    Ba-133    16.20 kBq on 07.06.2010    MEYR 197/R-002
3687RP    Eu-152    20.64 kBq on 07.06.2010    MEYR 197/R-002
```

It is possible to specify the activities using the *-a activity_133Ba:activity_152Eu* switch and the time at which they were determined using the *-t unix_time_of_activity*. It is also possible to specify the measurement time (the default value is the current time) with the *-m unix_time_of_measurement*. The unix times are the number of seconds since 1970. You can use the command *date* with the -d option to specify the date and +%s to perform the conversion. e.g.

```
analyse_EuBa -a16.2:20.64 -t$(date -d "7 Jun 2010" +%s) -m$(date -d "7 Jul 2010" +%s)
```

which indicates the ^{133}Ba source had 16.2 kBq and the ^{152}Eu source 20.64 kBq on 7 Jun 2010 and the measurement was performed a month later.

Note that as we only use the ratio between these two activities in the code, the units don't matter as long as the same units are used for both sources.

If you are working on a slow connection, you might find it useful to use the *-b* switch which turns on batch mode. Then, the output files are all generated, including the .pdf and .png copies of the TCanvas, but the TCanvas is not actually displayed.

9 The positioning code

In order to determine the positions of the Miniball detectors, we take data with a ^{22}Ne beam on a deuterated polyethylene target and look at the 440 keV peak from the d(^{22}Ne, ^{23}Na)n reaction, using its doppler shifted energy for each segment to determine the angles.

We use the program *sort_22Ne* to sort the data and then *analyse_20Ne* to determine the energies of the doppler-shifted 440 keV peaks for each segment. The next step is to compare these energies to the calculated ones for a given set of r, θ, ϕ, α and β, which we then adjust.

The problem is that we have four values for each cluster (r, θ, ϕ and α) and one value common to all eight clusters (β) so we are trying to find a minimum in an 33-dimensional space, which has lots of local minima.

There is no way to automate this procedure sensibly, so it has to be done by hand. So instead, a root class called *PositionClusters* has been created. You need to create an instance of this class:

```
PositionClusters p;
```

and then you can use it.

9.1 Setting up a scan

The command *Setup* is used from the *PositionClusters* class to set up the scan. e.g.:

```
p.Setup(12, 156.3, 52.3,  313.9, 300.5, 0.0667); // Forward left up
```

This indicates that we are working with cluster 12 (this determines the set of energies it reads from the *analyse_22Ne.dat* file, that the starting value of r is

156.3 mm, of θ is 52.3°, of ϕ is 313.9°, α is 300.5° and β is 0.0667.

The code will then create a TCanvas for this cluster divided up into a 2 x 3 grid. It then scans each parameter in turn around the specified values, always leaving the other values at the values set by the user. It plots the χ^2 as a function of that parameter. In the sixth TPad of the grid it displays the results.

9.2 Saving the canvas

The command *SaveCanvas* is used from the *PositionClusters* class save the canvas as both .pdf and .png.

```
p.SaveCanvas();
```

The name for the files is generated based on the cluster number, so cluster 16 will generate names *analyse_22Ne_16.pdf* and *analyse_22Ne_16.png*.

9.3 Writing the results

The command *WriteResults* is used from the *PositionClusters* class to write the results for all the clusters to a *config.dat* file in the format used by offl_root_med.

```
p.WriteResults();
```

9.4 Tips

Note that ϕ is the one parameter you can actually read off the frame accurately. The value of θ can be obtained within a few degrees by getting the accurate position of the arm and adding or subtracting about 18.5° offset between the arm and the centre of the cluster. The value of α has to be estimated by eye. If $\alpha = 0°$ it means A3 is pointing upwards, $\alpha = 120°$ means C3 is pointing upwards and $\alpha = 240°$ means B3 is pointing upwards.

10 The program medsort

The program medsort is a set of modules for sorting Miniball Event Data (MED) files for debugging purposes. It is not for analysis. The actualy *medsort* command is, in fact, a link to *modcode* which is a general purpose module loading code, that does nothing, but can load modules to do things. So what it does depends on the modules you load.

The behaviour is different when *modcode* is called as *modcode* or as *medsort* because it uses the name to build the name of environment variable which should point to the location of the modules. So to use *medsort* you need to have an environment variable *modcode_medsort* defined to point to the directory containing the .so files belonging to *medsort*.

In order to run *medsort* you need to provide the name of a module with the *-m* option. You can either specify a .so file with its full path, or just the base name, in which case it looks in the directory pointed to by *modcode_medsort*.

Some modules require variables to be set and this can be done using the *-v* option. e.g. if you want to set the variable MAXBUF to 1000 you would do -v MAXBUF=1000.

Then you need to give a list of one or more MED files on the command line. Sometimes you want to pass switches to modules. This leads to the issue of whether a switch is intended for *medsort* or for the underlying module. To make this clear, use the – notation. Everything before the double-minus-sign is a switch for *medsort* and everything after for the modules.

You can get a list of modules with:

```
medsort -h
```

10.1 The modules

10.1.1 show_mbs_timestamp

This module shows the MBS timestamp which is based on the system time of the power PC during acquisition. However, it includes microseconds as well as the normal unix time.

Usage:

```
medsort -m show_mbs_timestamps *.med
```

which gives an output like:

```
Timestamp 0       : Fri 09Jul2010 09:25:49.062443848
Timestamp 4809    : Fri 09Jul2010 09:26:49.001068121
Timestamp 9331    : Fri 09Jul2010 09:27:49.009767333
Timestamp 13855   : Fri 09Jul2010 09:28:49.021701181
```

10.1.2 show_dgf_buffer_headers

This module shows the DGF buffer header information. i.e. for each DGF buffer in the MED file, the number of words, format, module number and timestamp

of the start of the acquisition of that buffer.

Usage:

```
medsort -m show_dgf_buffer_headers *.med
```

which gives an output like:

```
B    0: Nwords=  66 Module=   3 Format=0x0101 Timestamp = 0x00000000FEC1
B    1: Nwords=  84 Module=   4 Format=0x0101 Timestamp = 0x00000000FEBF
B    2: Nwords=  66 Module=   9 Format=0x0101 Timestamp = 0x00000000FEC0
B    3: Nwords=  84 Module=  10 Format=0x0101 Timestamp = 0x00000000FEC0
B    4: Nwords=  36 Module=  13 Format=0x0101 Timestamp = 0x00000000FEBF
```

10.1.3 show_dgf_event_headers

This module shows the DGF event header information. i.e. for each DGF event in the MED file, the hit pattern and the even timestamp. Since this information is hard to interpret without the buffer header, this module implicitly loads the *show_dgf_buffer_headers* module and shows both the buffer and event headers.

Usage:

```
medsort -m show_dgf_event_headers *.med
```

which gives an output like:

```
B0: Nwords=66 Module=3 Format=0x0101 Timestamp = 0x00000000FEC1
        E0: Pattern=0x0007 Timestamp=0x0000000694E5 (9152.900 us)
        E1: Pattern=0x0007 Timestamp=0x000000069D34 (9206.075 us)
B1: Nwords=84 Module=4 Format=0x0101 Timestamp = 0x00000000FEBF
        E0: Pattern=0x000F Timestamp=0x0000000694E4 (9152.925 us)
        E1: Pattern=0x000F Timestamp=0x000000069D34 (9206.125 us)
B2: Nwords=66 Module=9 Format=0x0101 Timestamp = 0x00000000FEC0
        E0: Pattern=0x0007 Timestamp=0x00000006B593 (9362.075 us)
        E1: Pattern=0x0007 Timestamp=0x00000006C6A3 (9471.275 us)
```

10.1.4 show_dgf_channel_headers

This module shows the DGF channel header information. i.e. for each DGF channel in each DGF event event in the MED file, the energy and trigger time. Since this information is hard to interpret without the event and buffer header, this module implicitly loads the *show_dgf_event_headers* module which in turn loads *show_dgf_buffer_headers*and shows the buffer, event and channel headers.
Usage:

```
medsort -m show_dgf_channel_headers *.med
```

which gives an output like:

```
B0: Nwords=66 Module=3 Format=0x0101 Timestamp = 0x00000000FEC1
        E0: Pattern=0x0007 Timestamp=0x0000000694E5 (9152.900 us)
                3.0 E=1363  T=0x0000000694FA
                3.1 E=1348  T=0x0000000694FA
                3.2 E=0     T=0x0000000694FA
        E1: Pattern=0x0007 Timestamp=0x000000069D34 (9206.075 us)
                3.0 E=2365  T=0x000000069D4B
                3.1 E=2346  T=0x000000069D4B
                3.2 E=0     T=0x000000069D4B
B1: Nwords=84 Module=4 Format=0x0101 Timestamp = 0x00000000FEBF
        E0: Pattern=0x000F Timestamp=0x0000000694E4 (9152.925 us)
                4.0 E=0     T=0x0000000694FA
                4.1 E=0     T=0x0000000694FA
                4.2 E=0     T=0x0000000694FA
                4.3 E=0     T=0x0000000694FA
        E1: Pattern=0x000F Timestamp=0x000000069D34 (9206.125 us)
                4.0 E=0     T=0x000000069D4B
                4.1 E=0     T=0x000000069D4B
                4.2 E=0     T=0x000000069D4B
                4.3 E=0     T=0x000000069D4B
```

10.1.5 show_vme_scalers

This module shows the VME scalers each time the occur in the data stream.

Usage:

```
medsort -m show_vme_scalers *.med
```

which gives an output like:

```
0  PPAC X1              0          1  PPAC X2              0
2  PPAC X3              0          3  PPAC X4              0
4  PPAC X5              0          5  PPAC X6              0
6  PPAC X7              0          7  PPAC X8              0
8  PPAC X9              0          9  PPAC X10             0
10 PPAC X11             0          11 PPAC X12             0
12 PPAC X13             0          13 PPAC X14             0
14 PPAC X15             0          15 PPAC X16             0
16 PPAC X17             0          17 PPAC X18             0
18 PPAC X19             0          19 PPAC X20             0
```

```
20 PPAC X21             0          21 PPAC X22             0
22 PPAC X23             0          23 PPAC X24             0
24 PPAC X25             0          25 PPAC X26             0
26 PPAC X27             0          27 PPAC X28             0
28 PPAC X29             0          29 PPAC X30             0
30 PPAC X31             0          31 PPAC X32             0
32 PPAC Y1              0          33 PPAC Y2              0
34 PPAC Y3              0          35 PPAC Y4              0
36 PPAC Y5              0          37 PPAC Y6              0
38 PPAC Y7              0          39 PPAC Y8              0
40 PPAC Y9              0          41 PPAC Y10             0
42 PPAC Y11             0          43 PPAC Y12             0
44 PPAC Y13             0          45 PPAC Y14             0
46 PPAC Y15             0          47 PPAC Y16             0
48 PPAC Y17             0          49 PPAC Y18             0
50 PPAC Y19             0          51 PPAC Y20             0
52 PPAC Y21             0          53 PPAC Y22             0
54 PPAC Y23             0          55 PPAC Y24             0
56 PPAC Y25             0          57 PPAC Y26             0
58 PPAC Y27             0          59 PPAC Y28             0
60 PPAC Y29             0          61 PPAC Y30             0
62 PPAC Y31             0          63 PPAC Y32             0
64 Q1 free              0          65 Q2 free              2
66 Q3 free              0          67 Q4 free              1
68 Q1 delayed           0          69 Q2 delayed           2
70 Q3 delayed           0          71 Q4 delayed           1
72 Q1 accepted          0          73 Q2 accepted          2
74 Q3 accepted          0          75 Q4 accepted          1
76 Q1 and gamma         0          77 Q2 and gamma         0
78 Q3 and gamma         0          79 Q4 and gamma         1
80 Q1 gate              0          81 Q2 gate              0
82 Q3 gate              0          83 Q4 gate              0
84 EBIS pulse           38         85 Total DGF            7321
86 Si OR                0          87 SYNCH                76
88 GFLT                 76         89 1 MHz                1000000
90 1 MHz and on win     30377      91 1 MHz and GFLT       60764
92 T1                   1          93 PS                   0
94 1MHz and laser on    0          95 Bragg                0
```

This repeats once for each second of data.

10.1.6 show_vme_scalers_average

This module shows the VME scalers for the free particle trigger rate averaged per second over every five minutes of acquisition time.

Usage:

```
medsort -m show_vme_scalers_average *.med
```

which gives an output like:

```
Thu 19Aug2010 16:26:22          127.27     131.24     157.81     145.74
Thu 19Aug2010 16:31:22          131.07     133.17     161.95     148.60
Thu 19Aug2010 16:36:22          135.86     139.00     164.41     152.08
```

10.1.7 show_sum_of_vme_scalers

This module shows the sum of each VME scaler for the whole file. As it goes through the data it gives a message periodically to show how many triggers it has treated and at the end it shows the scalers.

Usage:

```
medsort -m show_sum_of_vme_scalers *.med
```

which gives an output like:

```
PPAC X1             0          PPAC X2             0
PPAC X3             0          PPAC X4             0
PPAC X5             0          PPAC X6             0
PPAC X7             0          PPAC X8             0
PPAC X9             0          PPAC X10            0
PPAC X11            0          PPAC X12            0
PPAC X13            0          PPAC X14            0
PPAC X15            0          PPAC X16            0
PPAC X17            0          PPAC X18            0
PPAC X19            0          PPAC X20            0
PPAC X21            0          PPAC X22            0
PPAC X23            0          PPAC X24            0
PPAC X25            0          PPAC X26            0
PPAC X27            0          PPAC X28            0
PPAC X29            0          PPAC X30            0
PPAC X31            0          PPAC X32            0
PPAC Y1             0          PPAC Y2             0
PPAC Y3             0          PPAC Y4             0
PPAC Y5             0          PPAC Y6             0
PPAC Y7             0          PPAC Y8             0
PPAC Y9             0          PPAC Y10            0
PPAC Y11            0          PPAC Y12            0
```

```
PPAC Y13            0          PPAC Y14            0
PPAC Y15            0          PPAC Y16            0
PPAC Y17            0          PPAC Y18            0
PPAC Y19            0          PPAC Y20            0
PPAC Y21            0          PPAC Y22            0
PPAC Y23            0          PPAC Y24            0
PPAC Y25            0          PPAC Y26            0
PPAC Y27            0          PPAC Y28            0
PPAC Y29            0          PPAC Y30            0
PPAC Y31            0          PPAC Y32            0
Q1 free             17102      Q2 free             7215
Q3 free             36537      Q4 free             12342
Q1 delayed          12117      Q2 delayed          4371
Q3 delayed          29975      Q4 delayed          7403
Q1 accepted         12117      Q2 accepted         4371
Q3 accepted         29975      Q4 accepted         7403
Q1 and gamma        601        Q2 and gamma        409
Q3 and gamma        1040       Q4 and gamma        590
Q1 gate             10784      Q2 gate             3247
Q3 gate             27174      Q4 gate             6228
EBIS pulse          99131      Total DGF           19360056
Si OR               47650      SYNCH               198249
GFLT                198249     1 MHz               2630000000
1 MHz and on win    79249023   1 MHz and GFLT      158508472
T1                  820        PS                  58
1MHz and laser on   1322395078 Bragg               0
```

10.1.8 show_mbs_triggers

This module shows MBS triggers. You get a trigger 14 at the start of the data and a trigger 15 at the end and a trigger 1 for each event. It is probably not that useful to use this module on its own, but if loaded together with another module, the trigger messages can be used to separate the events in the output.

Usage:

```
medsort -m show_mbs_triggers *.med
```

which gives an output like:

```
Trigger 14 (START)
Trigger 1 (EVENT)
Trigger 1 (EVENT)
Trigger 1 (EVENT)
Trigger 1 (EVENT)
```

10.1.9 show_patterns

This module shows the patterns from the pattern unit each time they occur in the MED file.

Usage:

```
medsort -m show_patterns *.med
```

which gives an output like:

```
000000C0 "Q4" "Q4 and gamma"

00000040 "Q4"

00000103 "Q1" "Q1 and gamma" "Laser"

00000001 "Q1"

000000C0 "Q4" "Q4 and gamma"
```

Note that "Laser" actually means "laser off" now, because the logic is inverted. i.e. the bit is zero for laser on and one for laser off.

10.1.10 show_subevents

This module shows the MBS subevent types.

Usage:

```
medsort -m show_subevents *.med
```

which gives an output like:

```
Start trigger 14
MBS timestamp Fri 09Jul2010 09:25:49.062443848
Dummy subevent [111,111]
Dummy subevent [111,111]
Dummy subevent [111,111]
End trigger 14
------------------------------
Start trigger 1
MBS timestamp Fri 09Jul2010 09:25:49.254938333
DGF buffer 32
DGF buffer 37
DGF buffer 38
```

```
DGF buffer 41
DGF buffer 42
DGF buffer 45
DGF buffer 46
DGF buffer 53
End trigger 1
------------------------------
```

10.1.11 show_number_of_events

This module shows the number of MBS events in the MED file.

Usage:

```
medsort -m show_number_of_events *.med
```

which gives an output like:

```
Opening AoQ4_009.med
Closed AoQ4_009.med with 198634 events
```

10.1.12 show_particle_rate

This module shows the particle rates in counts per second during the MED file.

Usage:

```
medsort -m show_particle_rates *.med
```

which gives an output like:

```
Period = 120 seconds
125.199054569   120.150 129.300 147.325 140.508  Mon 12Sep2016 23:24:20.841878909
245.199032144   119.608 127.200 149.283 142.208  Mon 12Sep2016 23:26:20.841856484
365.198684993   118.842 127.517 152.583 138.058  Mon 12Sep2016 23:28:20.841509333
485.198306447   120.908 129.142 152.883 141.433  Mon 12Sep2016 23:30:20.841130787
605.197933599   119.592 129.383 152.850 139.575  Mon 12Sep2016 23:32:20.840757939
```

It accepts the variable *period*, which selects the period over which the rates should be averaged. A negative value for *period* corresponds to that number of supercycles. e.g.

```
medsort -m show_particle_rates -v period=-5 *.med
```

averages over 5 supercycles.

10.1.13 show_proton_supercycle

This module shows the proton pulses in the supercycle.

Usage:

```
medsort -m show_proton_supercycle *.med
```

which gives an output like:

```
Timestamper module is number 53
                       - Thu 13Oct2016 15:52:36    1 * 1+x 3+x 6+x 8+x 9+x 11+x 12+x 15+x (8
Thu 13Oct2016 15:52:36 - Thu 13Oct2016 16:07:09   26 * 5 8 11 14 16 19 21 22 24 25 28 (11/28
Thu 13Oct2016 16:07:09 - Thu 13Oct2016 16:16:41   18 * 5 6 8 11 14 16 17 19 21 22 24 25 28 (
Thu 13Oct2016 16:16:41 -                           1 * 5 6 (2/28)
```

In this case, the first supercycle in the MED file is missing the first pulses. Here x is 13, so we have pulses 14, 16, 19, 21, 22, 24, 25 and 28. Then we have 26 supercycles with 11 pulses per supercycle and then at 4:07 PM, the supercycle was changed and we had 18 more supercycles with 13 pulses per supercycle. Finally, the last supercycle was truncated by the end of the MED file.

10.1.14 show_t1_ps

This module shows the proton pulses in the supercycle, much like *show_proton_supercycle* but it also shows the DGF timestamps.

Usage:

```
medsort -m show_t1_ps *.med
```

So the output looks like:

```
Thu 13Oct2016 16:08:50.746320181   PS   T=0x00098F2AAEA6  33.600
Thu 13Oct2016 16:08:56.426178606   T1   T=0x00099C995F52  5.634              5
Thu 13Oct2016 16:08:57.628284242   T1   T=0x00099F75C9A2  6.834              6
Thu 13Oct2016 16:09:00.026102909   T1   T=0x0009A52E9E3F  9.234              8
Thu 13Oct2016 16:09:03.626012484   T1   T=0x0009ADC3DD2B  12.834             11
Thu 13Oct2016 16:09:07.225897515   T1   T=0x0009B6591C17  16.434             14
Thu 13Oct2016 16:09:09.625859272   T1   T=0x0009BC11F0B3  18.834             16
Thu 13Oct2016 16:09:10.825838363   T1   T=0x0009BEEE5B01  20.034             17
Thu 13Oct2016 16:09:13.225754787   T1   T=0x0009C4A72FA0  22.434             19
Thu 13Oct2016 16:09:15.625706303   T1   T=0x0009CA60043C  24.834             21
Thu 13Oct2016 16:09:16.825684545   T1   T=0x0009CD3C6E8C  26.034             22
Thu 13Oct2016 16:09:19.225628848   T1   T=0x0009D2F54328  28.434             24
Thu 13Oct2016 16:09:20.425581212   T1   T=0x0009D5D1AD78  29.634             25
Thu 13Oct2016 16:09:24.025480303   T1   T=0x0009DE66EC64  33.234             28
```

10.1.15 show_repetition_rate

This module shows the EBIS frequency.

Usage:

```
medsort -m show_repetition_rate *.med
```

```
Timestamper module is number 53
Averaging over a period of 300 seconds
Sun 16Oct2016 17:44:05 79.99938 ms = 12.50010 Hz
Sun 16Oct2016 17:49:05 79.99925 ms = 12.50012 Hz
Sun 16Oct2016 17:54:05 79.99923 ms = 12.50012 Hz
Sun 16Oct2016 17:59:05 79.99922 ms = 12.50012 Hz
```

The period can be changed using the variable *period*.

10.1.16 show_on_off_window

This module shows the average length of the on and off windows and the EBIS frequency.

Usage:

```
medsort -m show_on_off_window *.med
```

which gives an output like:

```
On win:  799.437 us   Off win: 799.542 us  EBIS freq: 37.692 Hz
```

10.1.17 show_dgf_scalers

This module shows the DGF scalers each time they occur in the MED file.

Usage:

```
medsort -m show_dgf_scalers *.med
```

which gives an output like:

```
Module:     1
REALTIMEA       0         REALTIMEB       802       REALTIMEC       0         RUNTIMEA        0
RUNTIMEB        709       RUNTIMEC        57401     GSLTTIMEA       0         GSLTTIMEB       0
GSLTTIMEC       0         NUMEVENTSA      0         NUMEVENTSB      21        DSPERROR        0
SYNCHDONE       1         TEMPERATURE     0         BUFHEADLEN      6         EVENTHEADLEN    3
CHANHEADLEN     9         U14             0         USEROUT         24583     AOUTBUFFER      20476
LOUTBUFFER      8192      AECORR          17618     LECORR          44        ATCORR          0
```

```
LTCORR          0         HARDWAREID      1284      HARDVARIANT     1         FIFOLENGTH      4096
FIPPIID         11        FIPPIVARIANT    0         INTRFCID        0         INTRFCVARIANT   0
SOFTWAREID      519       SOFTVARIANT     4         LIVETIMEA0      0         LIVETIMEB0      44
LIVETIMEC0      26098     FASTPEAKSA0     0         FASTPEAKSB0     320       GSLTFPA0        0
GSLTFPB0        0         PUPEAKSA0       0         PUPEAKSB0       305       OVERFLOWA0      0
OVERFLOWB0      0         INSPECA0        0         INSPECB0        0         UNDERFLOWA0     0
UNDERFLOWB0     0         ADCPERDACA0     0         ADCPERDACB0     0         UNUSEDC0        0
LIVETIMEA1      0         LIVETIMEB1      44        LIVETIMEC1      26088     FASTPEAKSA1     0
FASTPEAKSB1     72        GSLTFPA1        0         GSLTFPB1        0         PUPEAKSA1       0
PUPEAKSB1       70        OVERFLOWA1      0         OVERFLOWB1      0         INSPECA1        0
INSPECB1        0         UNDERFLOWA1     0         UNDERFLOWB1     0         ADCPERDACA1     0
ADCPERDACB1     0         UNUSEDC1        0         LIVETIMEA2      0         LIVETIMEB2      44
LIVETIMEC2      26047     FASTPEAKSA2     0         FASTPEAKSB2     94        GSLTFPA2        0
GSLTFPB2        0         PUPEAKSA2       0         PUPEAKSB2       91        OVERFLOWA2      0
OVERFLOWB2      0         INSPECA2        0         INSPECB2        0         UNDERFLOWA2     0
UNDERFLOWB2     0         ADCPERDACA2     0         ADCPERDACB2     0         UNUSEDC2        0
LIVETIMEA3      0         LIVETIMEB3      44        LIVETIMEC3      26172     FASTPEAKSA3     0
FASTPEAKSB3     0         GSLTFPA3        0
```

10.1.18 show_dgf_livetime

This module shows the total live time for the DGFs for an MED file.

Usage:

```
medsort -m show_dgf_livetime *.med
```

which gives an output like:

```
Module:     1   99.96%  99.96%  99.96%  99.96%
Module:     2   99.96%  99.96%  99.96%  99.96%
Module:     3   99.96%  99.96%  99.96%  99.96%
```

for each scaler readout in the file.

10.1.19 show_sum_of_dgf_livetime

This module shows the total live time for the DGFs for an MED file.

Usage:

```
medsort -m show_sum_of_dgf_livetime *.med
```

which gives an output like:

```
Module: 1  Live:          33110.28297 s Run:          33125.01221 s 99.956 %
Module: 2  Live:          33113.59936 s Run:          33125.84647 s 99.963 %
```

```
Module: 3  Live:          33110.37250 s Run:          33124.96406 s 99.956 %
etc. etc.
Module: 53 Live:          33117.97390 s Run:          33122.88574 s 99.985 %
Module: 54 Live:          33096.33179 s Run:          33126.02318 s 99.910 %

Average    Live:          33109.62753 s Run:          33125.19628 s 99.953 %
```

10.1.20 show_particles_over_threshold

This module shows the particle count rates by counting events in the ADCs above a threshold and comparing with the MBS timestamp.

Usage:

```
medsort -m show_particles_over_threshold *.med
```

which gives an output like:

```
Fri 09Jul2010 09:30:49    0.007    0.017    0.000    0.033
Fri 09Jul2010 09:35:49    0.030    0.090    0.710    0.133
Fri 09Jul2010 09:40:49    0.163    0.350    4.607    1.007
Fri 09Jul2010 09:45:49    0.787    0.820    8.053    2.870
Fri 09Jul2010 09:50:49    1.153    0.197    1.547    2.617
Fri 09Jul2010 09:55:49    1.497    0.300    1.747    3.267
Fri 09Jul2010 10:00:49    1.900    0.330    2.013    4.347
Fri 09Jul2010 10:05:49    2.597    0.347    2.450    4.997
```

10.1.21 show_madc_data

This module shows the data from the MADC32 modules.

Usage:

```
medsort -m show_madc_data *.med
```

which gives an output like:

```
Module: 58 Timestamp: 0x00000046        30    30    28    29    34    32    33    36    28
Module: 58 Timestamp: 0x00000046        38    37    36    35    35    31    35    37    30
Module: 55 Timestamp: 0x00000046        60    31    36    27    32    32    35    35    34
Module: 55 Timestamp: 0x00000046        60    30    34    27    32    32    32    35    32
```

10.1.22 show_laser_on_off_fraction_from_scalers

This module shows the fraction of time that the laser was on in an MED file using the 1 MHz and laser on scaler vs. the 1 MHz scaler. Note that the signal only indicates the state of the request not what the laser actually does.

Usage:

```
medsort -m show_laser_on_off_fraction_from_scalers *.med
```

which gives an output like:

```
0.000              49.822 %
0.000              49.794 %
0.000              49.765 %
0.000              49.737 %
0.000              49.708 %
0.000              49.680 %
68.394             49.690 %
100.000            49.719 %
100.000            49.748 %
100.000            49.777 %
100.000            49.805 %
100.000            49.834 %
100.000            49.863 %
100.000            49.891 %
```

The left number is the current value and the right one is the average for the whole file. This run was in laser on/off mode.

10.1.23 show_laser_on_off_fraction_from_pattern_unit

This module shows the fraction of time that the laser was on in an MED file using the pattern unit bit. Note that the signal only indicates the state of the request not what the laser actually does.

Usage:

```
medsort -m show_laser_on_off_fraction_from_pattern_unit *.med
```

which gives an output like:

```
Trigger 3000   ; laser on = 1     ; laser_off = 2     ; ratio =       33.333 %
Trigger 6000   ; laser on = 1     ; laser_off = 3     ; ratio =       25.000 %
Trigger 9000   ; laser on = 1     ; laser_off = 4     ; ratio =       20.000 %
Trigger 12000  ; laser on = 3     ; laser_off = 4     ; ratio =       42.857 %
```

```
Trigger 15000  ; laser on = 3     ; laser_off = 8     ; ratio =       27.273 %
Trigger 18000  ; laser on = 4     ; laser_off = 8     ; ratio =       33.333 %
Trigger 21000  ; laser on = 4     ; laser_off = 8     ; ratio =       33.333 %
Trigger 24000  ; laser on = 5     ; laser_off = 8     ; ratio =       38.462 %
Trigger 27000  ; laser on = 5     ; laser_off = 8     ; ratio =       38.462 %
Trigger 30000  ; laser on = 6     ; laser_off = 8     ; ratio =       42.857 %
Trigger 33000  ; laser on = 7     ; laser_off = 8     ; ratio =       46.667 %
```

The left number is the current value and the right one is the average for the whole file. This run was in laser on/off mode.

10.1.24 is_laser_on

This module checks if the laser was in ON, ON/OFF or OFF mode.
Usage:

```
medsort -m is_laser_on *.med
```

which gives an output like:

```
IS557_065_78Zn_208Pb_pos2.med laser ON
IS557_066_78Zn_208Pb_pos2.med laser ON
IS557_067_78Zn_196Pt_pos1.med laser ON
IS557_068_78Zn_196Pt_pos1.med laser OFF
IS557_069_78Zn_208Pb_pos2.med laser ON
```

10.1.25 show_laser_power

This module is obsolete. In the past, the laser people provided us with an analogue signal "laser power" which we fed to one of the ADCs. This is no longer the case. Use the laser vistar instead.
Usage:

```
medsort -m show_laser_power *.med
```

10.1.26 show_number_of_bragg_events

This module shows the number of events from the Bragg chamber in an MED file.

Usage:

```
medsort -m show_number_of_bragg_events *.med
```

which gives an output like:

```
1024 Bragg events
```

10.1.27 scope_bragg

This module reads the Bragg data and sends the traces to the *scope* program. It gives no useful output to the screen.

Usage:

```
scope 0 &
scope 1 &
medsort -m scope_bragg *.med
```

10.1.28 scope_cd

This module is for displaying the 2D hitpattern of the CD in the Coulex configuration. It decodes the MUX data to plot ring vs. sector. It sends the data to the *scope2d* program. It gives no useful output to the screen.

Usage:

```
scope2d &
medsort -m scope_cd *.med
```

It can also use the *livedata* module to try to give a live display.

10.1.29 scope_cd_crex

This module is for displaying the 2D hitpattern of the CD in the C-REX configuration. It decodes the MUX data to plot ring vs. sector. It sends the data to the *scope2d* program. It gives no useful output to the screen.

Usage:

```
scope2d &
medsort -m scope_crex_cd *.med
```

It can also be used together with the *livedata* module to try to give a live display.

10.1.30 show_caen_data

This module shows the data from the CAEN modules.

Usage:

```
medsort -m show_caen_data *.med
```

which gives an output like:

```
crate = 00 module = 62:                3234
crate = 00 module = 62:           269
```

The horizontal position indicates which channel the event came from.

10.1.31 check_synch_vs_gflt

This module shows the number of SYNCs per second and the number of GFLTs per second. These values should be equal. If they are not, something is wrong with the electronics.

Usage:

```
medsort -m check_synch_vs_gflt *.med
```

which gives an output like:

```
Fri 09Jul2010 09:30:50          SYNCH=75.372093       GFLT=75.372093       ERROR=0.000000
Fri 09Jul2010 09:35:51          SYNCH=75.385382       GFLT=75.385382       ERROR=0.000000
Fri 09Jul2010 09:40:52          SYNCH=75.375415       GFLT=75.375415       ERROR=0.000000
Fri 09Jul2010 09:45:53          SYNCH=75.382060       GFLT=75.385382       ERROR=-0.003322
Fri 09Jul2010 09:50:54          SYNCH=75.382060       GFLT=75.378738       ERROR=0.003322
Fri 09Jul2010 09:55:55          SYNCH=75.382060       GFLT=75.382060       ERROR=0.000000
Fri 09Jul2010 10:00:56          SYNCH=75.382060       GFLT=75.382060       ERROR=0.000000
Fri 09Jul2010 10:05:57          SYNCH=75.375415       GFLT=75.375415       ERROR=0.000000
```

10.1.32 livedata

This module tries to throttle the reading of the MED file if it reaches a timestamp which is only a few seconds old. It can be used on the file which is currently being acquired and it tries to stay just a little behind the acquisition, so the data are more or less live. Unfortunately, there is no simple way to tell the difference between the end of a file because the DAQ has closed it and the end of the file, because the DAQ is waiting for the next event to write to it. So sometimes it will quit early.

The module makes no sense on its own and must be combined with other modules which provide useful output regularly, while parsing the file. e.g. with *show_particle_rate*:

```
medsort -m show_particle_rate -m livedata *.med
```

10.1.33 current

This module is used instead of providing the name of a .med file on the command line. It looks for the most recent .med file in the current working directory and passes it to the *treat_med* module. When that file is done, it looks to see if a new file has been created and if so treats it. If not, it waits until one is created.

On its own, it does nothing, but it can be combined with other modules to keep treating the current file.
e.g. we can run *show_particle_rate* on the current .med file and try to keep just behind the acquisition using *livedata*:

```
medsort -m show_particle_rate -m livedata -m current *.med
```

Note, that the logic for this is not quite perfect, since there is no way to know if there is no data in a file, because it was stopped short, or because it has yet to be written. So sometimes it repeats a whole file.

10.1.34 generate_dgf_spectra

This module generates spectra of the DGF data which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware gf2 format. It doesn't generate any useful output to the screen.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m generate_dgf_spectra *.med
```

10.1.35 generate_madc_spectra

This module generates spectra of the MADC32 data which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware gf2 format. It doesn't generate any useful output to the screen.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m generate_madc_spectra *.med
```

10.1.36 generate_caen_spectra

This module generates spectra of the CAEN data which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware gf2 format. It doesn't generate any useful output to the screen.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m generate_caen_spectra *.med
```

10.1.37 generate_particle_ISOLDE_spectra

This module generates spectra of the times of particles vs. EBIS, T1 and PS signals from ISOLDE, which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware

gf2 format. It doesn't generate any useful output to the screen.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m generate_particle_ISOLDE_spectra *.med
```

10.1.38 generate_particle_gamma_spectra

This module generates spectra of the times of particles vs. gamma, which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware gf2 format. It doesn't generate any useful output to the screen.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m generate_particle_gamma_spectra *.med
```

10.1.39 generate_number_of_events_spectra

This module generates spectra showing the distribution of the number of events per DGF buffer, which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware gf2 format. It doesn't generate any useful output to the screen.

Usage:

```
medsort -m generate_number_of_events_spectra *.med
```

10.1.40 generate_gamma_ISOLDE_spectra

This module generates spectra of the times of gammas vs. EBIS, T1 and PS signals from ISOLDE, which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware gf2 format. It doesn't generate any useful output to the screen.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m generate_gamma_ISOLDE_spectra *.med
```

10.1.41 extract_IC

This module generates of the energy loss in the gas and the energy in Si for the ionisation chamber, which are written into the directory pointed to by the environment variable *SW_HIST_DIR*. The spectra are written in the Radware

gf2 format. It doesn't generate any useful output to the screen.

It uses the variables *N_PER_SPEC* and *SEC_PER_SEC* to determine how many counts or how much time should be put in each spectrum. Values of zero (the default) mean there is no limit. If the value of one or both is non-zero, a new spectrum will be started after the specified number of events or seconds (of acquisition time).

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m extract_IC *.med
```

10.1.42 show_correlated_buffers

This module shows DGF buffers correlated by their timestamps. It shows the timestamp and the list of modules which had buffers matching that timestamp. Usage:

```
export SW_HIST_DIR=/tmp
medsort -m show_correlated_buffers *.med
```

which gives an output like:

```
00000000FEC0    3    4    5    6    17   18   19   20   21   22   23   24   25   26   29   3
0000000FE3F2    45   46   43   44
```

10.1.43 show_correlated_channels

This module seems to be broken. It should show the correlated channels.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m show_correlated_channels *.med
```

which gives an output like:

```
  29.0 0000000E2DB5 11725 1122
  29.1 0000000E2DB5 11725 1125
  29.2 0000000E2DB5 11725 0
  30.0 0000000E2DB5 11725 0
  30.1 0000000E2DB5 11725 0
  30.2 0000000E2DB5 11725 0
  30.3 0000000E2DB5 11725 0
--------------------
```

```
37.0 0000000E32F4 13062 1279
37.1 0000000E32F4 13062 0
37.2 0000000E32F4 13062 0
38.0 0000000E32F5 13062 0
38.1 0000000E32F5 13062 1342
38.2 0000000E32F5 13062 0
38.3 0000000E32F5 13062 0
```

The first number is the module and channel number, then the timestamp in hexadecimal, then the trigger time and finally the energy.

10.1.44 med_treat_file

This module treats an MED file. You must load it from any code that needs to set up callbacks for opening and closing files or for trigger callbacks directly. You should not load it from the command line.

10.1.45 med_treat_subevent

This module treats a subevent of an MED file. You must load it from any code that needs to set callbacks to unpack particular subevents. You should not load it from the command line.

10.1.46 med_unpack_bragg

This module contains the code for unpacking Bragg data. It should be loaded from any code that needs to treat Bragg data, but not loaded from the command line directly.

10.1.47 med_unpack_caen

This module contains the code for unpacking CAEN data. It should be loaded from any code that needs to treat CAEN data, but not loaded from the command line directly.

10.1.48 med_unpack_dgf_scalers

This module contains the code for unpacking DGF scaler data. It should be loaded from any code that needs to treat DGF scaler data, but not loaded from the command line directly.

10.1.49 med_unpack_dgf

This module contains the code for unpacking DGF data. It should be loaded from any code that needs to treat DGF data, but not loaded from the command line directly.

10.1.50 med_unpack_madc

This module contains the code for unpacking MADC data. It should be loaded from any code that needs to treat MADC data, but not loaded from the command line directly.

10.1.51 med_unpack_mbs_timestamp

This module contains the code for unpacking MBS timestamp data. It should be loaded from any code that needs to treat MBS timestamp data, but not loaded from the command line directly.

10.1.52 med_unpack_pattern

This module contains the code for unpacking pattern unit data. It should be loaded from any code that needs to treat pattern unit data, but not loaded from the command line directly.

10.1.53 med_unpack_vme_scalers

This module contains the code for unpacking VME scaler data. It should be loaded from any code that needs to treat VME scaler data, but not loaded from the command line directly.

10.1.54 ascii

This is a module to write ascii format spectra, either with just the contents of each channel on a separate line, or with the channel number and its contents on each line. It should not be loaded from the command line, but should be loaded by any module that uses either the *ascii_write* or the *ascii2_write* functions.

10.1.55 gf2

This is a module to write Radware gf2 format spectra. It should not be loaded from the command line, but should be loaded by any module that uses the

gf2_write function.

10.1.56 cracow

This is a module to write GSI Cracow format spectra. It should not be loaded from the command line, but should be loaded by any module that uses the *cracow_write* function.

10.1.57 raw

This is a module to write little endian raw format spectra either as short integers, long integers or 4-byte floating point. It should not be loaded from the command line, but should be loaded by any module that uses one of the *raw_short_write*, *raw_long_write* or *raw_float_write* functions.

10.1.58 spectrum

This is a module to write spectra in one of the supported formats. The environment variable *SW_HIST_DIR* is used to determine where the spectra should be written and the variable *SW_HIST_MODE* is used to chose the kind of spectrum to write. Valid values for this variable are: "GF2", "ASCII", "ASCII2", "RAW_FLOAT", "RAW_SHORT", "RAW_LONG" and "CRACOW". It should not be loaded from the command line, but should be loaded by any module that uses the *increment_spectrum* function.

It provides two functions:

- int write_all_spectra();
- int increment_spectrum(char *key, uint16_t module, int channel, uint16_t energy);

The spectra are allocated dynamically and the filenames are based on the combination of the key, the module and the channel number and are written in the *SW_HIST_DIR*. The *write_all_spectra* function may be called at any time, and will automatically be called when the program terminates. It writes any histograms which have been incremented.

10.1.59 command_line

This is a module to handle the command line passed to *medsort*. It should not be loaded from the command line, but only used in code that needs to be called whenever a file is treated.

10.1.60 IC_to_root

This is a module to generate a root tree of the ionisation chamber data. It also includes the timestamps for the EBIS, T1 and PS signals, so the ionisation chamber data can be compared to these events.

The variables *dE_offset*, *dE_slope*, *dE_quad* and *Erest_offset*, *Erest_slope*, *Erest_quad* are used to set the calibration for the Δ-E and E_{rest} values. If you set the variable *root*, the output root file will have this name.

You should load *libIC.so* in root in order to read such data. This root library is part of the *medsort* package and is installed in the root library directory. Note that if you upgrade root or move it around, you will need to rebuild it or copy it. To use it:

```
root
   gSystem->Load("libIC");
```

10.1.61 show_adc_tdc_mismatches

This is a module to check for mismatches between the CAEN ADC data and the CAEN TDC data. It is probably no longer useful.

10.1.62 correlate_buffers

This module correlates DGF buffers together based on their timestamp. It shouldn't be called directly from the command line, but should be used by modules which need to process correlated buffers.

10.1.63 correlate_channels

This module correlates DGF channel data together based on their timestamp. It shouldn't be called directly from the command line, but should be used by modules which need to process correlated channels.

10.1.64 calibrate

This module does not yet serve a useful purpose. The aim is to use it to calibrate the DGF data in order to do things like add back etc. However, for the moment, there is nothing which can use the calibrated data. So you should not use this module.

10.1.65 check_dgf_scaler_timestamper_3

This module was written for some specific purpose. It takes looks at the DGF scalers for the third timestamper DGF and counts the fast peaks in channel 0

(the timestamper DGFs only use this channel) and the total number of events. Every five minutes worth of data, it writes the values to standard output. Usage:

```
export SW_HIST_DIR=/tmp
medsort -m check_dgf_scaler_timestamper_3 *.med
```

which gives an output like:

```
Tue 13Jul2010 14:24:43 10.753333       10.873333
Tue 13Jul2010 14:29:43 2.876667        13.726667
Tue 13Jul2010 14:34:43 10.326667       24.326667
Tue 13Jul2010 14:39:43 17.496667       41.983333
```

First comes the timestamp for the data, then the number of events per second and finally the number of fast peaks per second.

10.1.66 generate_12C1_vs_time

This module was written for some specific purpose. It is unlikely to be useful to anyone. It generates three spectra from channel 12C1, which are written into the directory pointed to by the *SW_HIST_DIR* environment variable) for the first 50000 events, the next 35000 events and the rest of the events. This was for checking what happened to that channel when it failed.

Usage:

```
export SW_HIST_DIR=/tmp
medsort -m generate_12C1_vs_time *.med
```