

# FCal Test Beam DAQ: description of raw data file format

P.Gorbunov (Univ. of Toronto and ITEP, Moscow)

Version 1.8 (15 June 2005)<sup>1</sup>

Document source: [http://atlas-fcaltb.web.cern.ch/atlas-fcaltb/Memos/DAQ/...](http://atlas-fcaltb.web.cern.ch/atlas-fcaltb/Memos/DAQ/)

C-source codes referred to in the text can be found in pfcalt01:/home/daq/DAQ/ (src, lib, include)

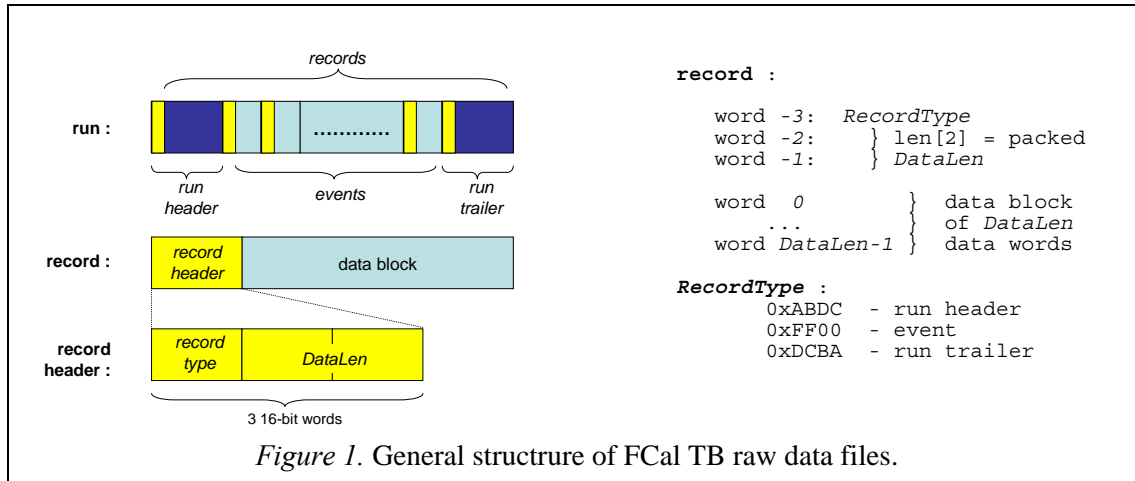
## Table of contents

<b>1</b>	<b>FCAL TB Data format overview.....</b>	<b>2</b>
1.1	General structure of the data files .....	2
1.2	Data blocks .....	2
1.2.1	Run header and trailer records.....	3
1.2.2	Event records.....	3
1.2.3	Event header sub-block .....	3
1.2.4	An example of an event hexdump .....	6
1.3	Utilities.....	7
1.3.1	rd_run, to dump the contents of a data file .....	7
<b>2</b>	<b>Run Configuration and Control keys.....</b>	<b>8</b>
2.1	Keys from the run...par (conditions) file.....	9
2.2	Keys from the cal...par file.....	11
2.3	Keys added by the online software .....	11
2.4	Keys related to the trigger grading.....	12
2.5	Keys related to the front-end electronics .....	12
2.6	Keys related to CAMAC.....	13
2.7	Keys referring to the configuration files .....	14
<b>3</b>	<b>Event sub-blocks .....</b>	<b>15</b>
3.1	FEB Data.....	15
3.1.1	The sub-block structure .....	15
3.1.2	Special considerations for auto-gain runs.....	17
3.2	Beam chambers data .....	17
3.2.1	The sub-block structure .....	17
3.2.2	A brief description of the BPCs and how to interpret the BPC data.....	18
3.2.3	BPC-related Run Header keys.....	19
3.3	Beam detectors, Tail Catcher, TIME data.....	20
3.3.1	Beam detectors sub-block.....	20
3.3.2	Tail Catcher sub-block .....	20
3.3.3	Time sub-block: id 0xFF03 .....	20
3.3.4	Notes about the CAMAC modules.....	21
3.4	Calibration Board Stamp.....	21
3.4.1	The sub-block structure .....	21
3.4.2	How to decode the channel pattern.....	22
	Appendix 2-A: an example of a complete run header block .....	23
	<b>Appendix 2-B: rhlib package, to handle the run-header records.....</b>	<b>24</b>
	<b>Appendix 3-A: An example of the FEB-data unpacking code.....</b>	<b>26</b>
	<b>Appendix 3-B FEB- and miniROD-related run header keys .....</b>	<b>27</b>
	<b>References.....</b>	<b>29</b>

<sup>1</sup> A (hopefully) better description of the FEB channel mapping (Appendices 3A and 3B)

## 1 FCAL TB Data format overview

### 1.1 General structure of the data files



The structure of the FCAL test beam data had been proposed by P. Loch (Ref.[1-1]) and is illustrated in Fig. 1. One *run* makes one binary raw data file, which is a contiguous sequence of variable length *records* ("blocks") of short 16-bit words. All records have the same logical structure, with a 3-word *header*<sup>2</sup> followed by a variable length *data block*. The record header contains the *record type* and the *data block length*.

The first record in every run is a *run header*, the last one is a *run trailer*, all other records are *events*. Respectively, there are only three different record types: 0xABDC, 0xFF00 and 0xDCBA.

The number of data words in a record is a 32-bit number *DataLen*, packed in two short words *len[2]* of the record header, as follows:

$$\text{DataLen} = (\text{long}) \text{len}[1] \mid (\text{long}) \text{len}[0] \ll 16 ;$$

The *DataLen* representation is encapsulated in DAQ functions `add_blk_header` and `get_blk_header`, also available for offline and monitoring applications as part of `my_event.c` package.

### 1.2 Data blocks

The data part of the records depends on the record type and may consist of items of type `char[]` (strings), `short` and `int`. If the file is read under Linux then (hopefully!) the numerical items should not require a byte-swapping.

<sup>2</sup> In the original event format specification, the block header consisted of 2 short words, with a 16-bit *DataLen*. This was changed to accommodate very long events (with 32 samples and >1 gain).

## 1.2.1 Run header and trailer records

The run header and trailer are, respectively, the first and the last records in any data file.<sup>3</sup>

The data part in these records is a block of plain ASCII text consisting of 64-character long *key-records* describing the run settings, DAQ configuration, and beam conditions (see Chapter 2 for further details). In brief, the run header can be considered as a portable run-related data base attached to the raw data file.

Run trailers are almost identical to the corresponding run headers and differ from the latter only by the presence of

```
RunStopDate
RunStopTime
Events
```

keys (Section 2.3).

The online `rhlib` package can be, optionally, used in offline applications to retrieve and manipulate the run header information (Chapter 2 and Appendix 2-B).

## 1.2.2 Event records

The data part in event records is logically split into a (variable) number of *sub-blocks* having the structure shown in Fig. 2, similar to the record structure: the sub-block starts with a 3-word header, followed by a data-part.

The first sub-block is always the *event header*, described in this Chapter. The event header is followed by a number of *data sub-blocks*, described in Chapter 3. The event composition (what kind of sub-blocks it contains) depends on the run type and event type. The order of sub-blocks is not fixed by the event format.

The sub-blocks in an event, like records in a file, follow each other without gaps, so one can iterate over sub-blocks by jumping from one sub-block header to another. However, there is a possibility to access sub-block directly, by using the *event directory* stored in the event header (sub-section 1.2.3.2). This redundancy is useful for event integrity checks.

## 1.2.3 Event header sub-block

### 1.2.3.1 The sub-block structure

```
SubBlockId : 0xFF01
Len        : 28
```

The event header data part, illustrated by Fig. 3, is described in sub-sections 1.2.3.2 and 1.2.3.3.

---

<sup>3</sup> Unless the run had finished abnormally (e.g., the DAQ program crashed or was killed by hand) – in which case the run header might be missing.



```

ev_dir[0] =IdFcal;      ev_dir[1] = offsetFcal ;
ev_dir[2] =IdTime;     ev_dir[3] = offsetTime ;
ev_dir[4] =IdTail;     ev_dir[5] = offsetTail ;
ev_dir[6] =IdBpc;      ev_dir[7] = offsetBpc ;
ev_dir[8] =IdBeam;     ev_dir[9] = offsetBeam;
ev_dir[10]=IdSlow;     ev_dir[11] = offsetSlow;
ev_dir[8] =0;          ev_dir[9] = 0;
ev_dir[10]=0;          ev_dir[11] = 0;
ev_dir[12]=0;          ev_dir[13] = 0;
ev_dir[14]=0;          ev_dir[15] = 0;
ev_dir[16]=IdEtc;     ev_dir[17] = offsetEtc ;
ev_dir[18]=IdCalStamp; ev_dir[19] = offsetCalStamp ;

```

Figure 4. Code showing how the event directory is filled in DAQ.

### 1.2.3.3 Other data words in the event header (event ID)

**ev\_number** (event header word 0)

**ev\_type** (event header word 1): 0=special; 1=physics; 2=FCal calibration (pulsar); 3=random (pedestal);  
4=BPC calibration

**ev\_time[2]** (event header words 2-3). Contain the readings from the 32-bit 10 MHz decoupler restarted at each Start-of\_Spill with the initial value 0xFFFFFFFF. Example of unpacking of the event time in Fortran:

```

integer*2 ev_time(2)
integer EvTime
equivalence (EvTime, ev_time)
...
* after reading of next event
ev_time=-1-ev_time ! the event time in 100 ns units

```

Same, in C:

```

unsigned int EvTime;
unsigned short ev_time[2];
...
// read next event
EvTime=0xffffffff-(ev_time[1]<<16 | ev_time[0]);

```

**ev\_trigger** (event header word 4) . Contains various trigger bits latched by CAMAC registers LeCroy 4448 and CEN 2047. See Fig. 5 for details. ). The signals are latched by the common busy of CIRQ which is delayed by ~70-80 ns wrt to the trigger.

**run\_number** (event header word 5)

**error** (event header word 7) if non-zero, indicates fatal read-out anomaly (or anomalies);

<i>LeCroy 4448 48-ch Coincidence register (N6 "Latch").</i>			
<i>Gate: L1 delayed</i>			
ch	0	- S1	ev_trigger bit 0
	1	- S2	1
	2	- S3	2
	3	- Veto	3
	4	- TC123 Sum	4
	5	- TC456 Sum	5
	6	- Muon	6
	7	- SFiber	7
			CIRQ input (1-6)
	8	- Beam trigger	2
	9	- Pedestal trigger	4
	10	- Calib trigger	5
			10
<i>PatternB CEN 2047 (N15 "Pile-up"). Gate: ??</i>			
ch	0	- Late Pileup Flag	11
	1	- Early Pileup flag	12
	2	- CEDAR 6/8	13
	3	- CEDAR 7/8	14
	4	- CEDAR 8/8	15

*Figure 5. The ev\_trigger word of the event header*

### flag (event header word 7)

```

bit 0 - undefined
bit 1 - out-of-spill flag (an OR of the software and hardware
        out-of-spill flags)
bit 2 - undefined
bit 3 - CIRQ start-of-spill bit
bit 4 - CIRQ beam trigger bit
bit 5 - CIRQ end-of-spill bit
bit 6 - CIRQ random (pedestal) trigger bit
bit 7 - CIRQ monitor (pulser) trigger bit
bit 8 - CIRQ BPC calib trigger bit

```

For example: flag=0x82 means "an out-of-spill" pulser trigger.

### 1.2.4 An example of an event hexdump

Data file: /raid/data/calib/cal550.dat

```

                                - EvLen -
0000900 .... .... .... ff00 0001 92ae ff01 0000 event header
0000910 001a 0001 0002 0000 0000 0000 0226 ff02
0000920 0029 ff03 0000 ff05 0000 ff06 0000 0000
0000930 0000 0000 0000 0000 0000 0000 0000 fffe
0000940 0000 ffff 001b ffff 0000 000b 0040 0000 id=0xFFFF
0000950 0000 0000 0000 0000 0000 0000 1f40 0000
0000960 0000 ff02 0001 9280 ffff ffff ffff ffff id=0xFF02
0000970 ffff ffff ffff ffff ffff ffff ffff ffff
0000980 ffff ffff ffff ffff 4303 4303 0302 0302
0000990 0301 0301 4300 4300 0307 0307 4306 4306
00009a0 4305 4305 0304 0304 0da3 0da3 0da3 0da3
00009b0 0da3 0da3 0da3 0da3 0da3 0da3 0da3 0da3
.....

```

## **1.3 Utilities**

The utility programs to manipulate data files are stored in /home/daq/Daq/bin directory, on pcfc101. The sources and .h files are in /home/daq/Daq/src and /home/daq/Daq/include, respectively.

### **1.3.1 rd\_run, to dump the contents of a data file**

```
rd_run <filename>
```

For example: `rd_run /raid/data/calib/cal600.dat`

This program is good for all data files (old and new format).

## 2 Run Configuration and Control keys

A *key-record* in FCal DAQ is just a text line beginning with an alphanumerical *keyword* followed by one or several *values*, all separated by any number of blanks. It can, optionally, contain a comment. Thus, there is nothing new or special about it: for example, Linux configuration files are based on a similar concept.

For FCalTB DAQ, key-records are important as

- a method of defining the run book-keeping/configuration information;
- a basis of the internal configuration data-base (DB);
- a method of storing the configuration/settings information (*meta-data* in ATLAS jargon) together with the raw data (in run headers, Section 1.2.1).

Having ASCII keyword-based configuration files makes it easy to modify the DAQ configuration and keep it readable, self-documented and compact. The run-settings generated by the run-control panel are also in this format.

If a new keyword with a correct formal syntax is added to any configuration file, it will be automatically copied to the run header and to the internal DAQ data base (DB). The value specified in this new key (or a list of values, for multi-value key-records) can be retrieved in the DAQ by a simple generic call to the internal DAQ DB.

The advantages of retaining this form for the run headers is that

- no special browser is required to see the run configuration for a given data file (e.g., one can use `od -c | more` or even just `more` to see quickly the run header contents);
- most importantly, it provides a natural solution to the test beam Conditions/Configuration data base problem: the relevant DB information is simply *attached* to the data itself, in a portable and flexible form, easily convertible to any post-data-taking data base. We were not bound to any specific DB specification during the data-taking, but our raw data is intrinsically DB-ready. On the other hand, the underlying online software is very light-weight and transparent, with only rudimentary internal DB functionality implemented (store, retrieve, dump).

The configuration and control information is read from several configuration files at a run start. All meaningful records (keys) from these files are copied to the run header as plain 64-byte long character strings (C language `char[64]` type). All keys appearing in FCal data files are described in this Chapter.

The syntax of a key-record is as follows:

```
keyword value [values..] [// comment]
```



The keyword // (double-slash) indicates "end-of-data". All records that follow are ignored.<sup>4</sup> A // in the middle of the record separates the value(s) from an optional comment. The keyword \* (asterisk) signifies a purely commentary line. For example:

```
* An example of a short configuration file
RunNumber 287
miniROD    1    2    3
FebAddr    0x22 0x28 0x31 // this is a comment
// -- this is end-of-data, followed by spare records
CalNtrig 1000
CalDac 1000
CalNpatt 1
CalPatt_0 0 7 15 21:47;8 90:127
```

The numerical values are either (signed) decimal integers, or hexadecimal numbers (0x....). Text values need not be quoted. Multivalued keys are detected automatically. The following syntax can be used to specify groups of numbers:

`n1:n2[:n3]`     *A range, for example*   0:127;16     12:34

`Ntimes*Value`     *A multiplier, for example*   20\*0

An example of a complete run header is given in Appendix 2-A.

Some limitations of the key-record syntax, namely:

- the keyword must start in the first character;
- no support for multi-line keys;
- the limited record length (64 characters),

were intentional, to keep the key-record handling codes (`rhlib` package, briefly described in Appendix 2-B) simple and robust. These limitations are compensated by a rich multi-value functionality.

## 2.1 Keys from the `run...par (conditions) file`

key	# of values	Default value(s)	Value type, description and range
<b>RunNumber</b>	1	-	int, <65535 <i>e.g. RunNumber 500</i>
<b>RunType</b>	1	1	int, 1=phys, 2=calib, 3=ped, 0 = special <i>e.g. RunType 2</i>
<b>DataStore</b>	2	0  none	V1=int, raw data destination 0=none, 1=file(directory), 2=dispatcher (@host) V2=text string <i>e.g. DataStore 1 /raid/data/calib</i> <i>DataStore 2 @pcfc101</i> <i>DataStore 0 none</i>

<sup>4</sup> This feature is used in the DAQ configuration files to hide spare records behind the //. Neither the // keyword, nor the records that follow are copied to the run header. Purely commentary lines (the ones starting with an \*) are also not copied to the run header. On the other hand, some commentary lines can be generated by the DAQ itself (see Section 2.3).

<i>key</i>	<i># of values</i>	<i>Default value(s)</i>	<i>Value type, description and range</i>
<b>BeamMomentum</b> (22/06/03)	1	0	int, beam momentum in [GeV], always positive <i>e.g. BeamMomentum 200</i>
<b>BeamParticle</b>	1	-1	int, the dominant beam particle, if known : 1 =e+, 2 =pi+, 3 =p+, 4 =mu+, 11=e-, 12=pi-, 14=mu-, (22/06/03) -1 unknown/irrelevant <i>e.g. BeamParticle 11</i>
<b>CryoX</b> <sup>5</sup>	1	0	int, the cryostat lateral (X) position, in [mm] <i>e.g. CryoX 25</i>
<b>CryoAngle</b> <sup>5</sup>	1	0	int, the cryostat angle, in [degrees] <i>e.g. CryoAngle 5</i>
<b>TableY</b>	1	0	int, the table vertical (Y) offset, in [mm] <i>e.g. TableY -18</i>
<b>BeamSpot</b>	1	-1	int, the beam spot position 1 = spot_1 in the Run Control panel 2 = spot_2        "-" 3 = spot_3        "-" 4 = spot_4U      "-" 5 = spot_4D      "-" -1 = undefined  "-"
<b>ReadOutMask</b>	1..10	All	text, list of data blocks to be recorded. All - all connected hardware Fcal - miniRODs Time Tail - tail catcher counters Bpc - wire chambers Soft - software data CalB - calibration board data for calib events XXXX=0 - to suppress block XXXX <i>e.g. ReadOutMask All Tail=0</i>
<b>miniROD</b>	1..8	0	int, miniRODs used. The read-out is done in the same order, as they are listed. <i>e.g. miniROD 1 3 4</i> <i>miniROD 1:8</i>
<b>Bpc</b>	1..6	0	int, BPCss used. The read-out is done in the same order, as they are listed. <i>e.g. Bpc 1 3</i> <i>Bpc 1:6 // all BPCs</i>
<b>MaxEvents</b>	1	1e10	int, Run limit condition by number of events. This is a soft condition. The actual number of events in a run can be larger, because DAQ never stops in the middle of a spill.
<b>MaxBursts</b>	1	1e10	int, Run limit condition by number of bursts.
<b>RunDebug</b>	15	all 0	int, Print level for different DAQ components 0 - no print 1 - a few prints per run 2 - a few prints per event 3 - a dozen prints per events 4 - prints too much (debug level)

<sup>5</sup> 22/06/03 obsolete, replaced with BeamSpot

## 2.2 Keys from the cal\_...par file

key	# of values	Default value(s)	Value type, description and range
<b>CalSDelays</b>	any	0	int, list of calibration pulse delay(s) in the corresponding calibration super-cycles, in [ns]. The CalSDelays key makes a CalDelays key void (ie, the latter will not be taken into account) <i>e.g. CalSDelays 0:17 // 18 super-cycles with constant delays 0,...,17</i>
<b>CalNtrig</b>	1	0	int, number of consecutive cal. pulses with the same parameters. <i>e.g. CalNtrig 1000</i>
<b>CalDac</b>	1..256	0	int, list of DAC values for each pattern <i>e.g. CalDac 1000:10000;500</i> <i>CalDac 0x12345</i>
<b>CalNpatt</b>	1	1	int, number of palibration patterns in one super-cycle.
<b>CalPatt_x</b>	1..128	0:127	int, list of FEB pins to be pulsed in the corresponding pattern "x" (x=0..127) (ie, CalPatt_0 - for pattern #0, CalPatt_1 - for pattern #2 etc) <i>e.g. CalPatt_14 0 7 15 21:47;8 90:127 // a weird pattern</i>
<b>CalDelays</b>	=CalNpatt	0	int, list of calibration pulse delays in [ns], for each of the requested patterns. This key is ignored if a CalSDelays key appears in the same configuration file. The number of values should be equal to the number of patters requested by CalNpatt key. <i>e.g. CalNpatt 20</i> <i>CalPatt_0 ...</i> <i>...</i> <i>CalPatt_20 ...</i> <i>CalDelays 0:10 9*11 // 0,1...10 ns, followed by 9 delays of 11 ns</i>

## 2.3 Keys added by the online software

<b>RunDate</b>	1	-	int, the run start date, as YYYYMMDD <i>e.g. RunDate 20030512</i>
<b>RunTime</b>	1	-	int, the run start time, as HHMMSS <i>e.g. RunTime 142501</i>
<b>RunStopDate</b>	1	-	int, the run stop date, as YYYYMMDD <i>e.g. RunDate 20030512</i>
<b>RunStopTime</b>	1	-	int, the run stop time, as HHMMSS <i>e.g. RunTime 142501</i>
<b>Events</b>	1	-	int, total number of event records in the run
<b>* pathname</b>	-	-	a purely commentary line containing the full pathname of the configuration file from which the following key-records are read <i>e.g. * ----- /raid/daq/config/par/cam/test_cam.par</i>

## 2.4 Keys related to the trigger grading

Define the trigger ratios during the spill and, optionally, disable main types of triggers. Value 0 means "disabled". The mechanism, proposed by A.Savin, works as follows. At start of run the software downcounters for Beam, Pedestal and Calibration triggers are preset with the values defined by the run header keys (see below). Upon reaching 0, the corresponding trigger is disabled until all other downcounters have reached zero values. Then whole cycle repeats. Between the spills, as many pedestal and calibration triggers are taken, as were during the previous spill. Thus, a zero preset value disables the corresponding trigger completely.

key	# of values	Default value(s)	Value type, description and range
TrigBeam	1	0	int, the preset value for "Beam" down-counter
TrigPed	1	0	int, same, for "Pedestal" (random) trigger
TrigFcal	1	0	int, same, for "Calibration" (monitor) trigger

## 2.5 Keys related to the front-end electronics

FebSamples	1	7	int, the number of samples recorded by FEBs. Range: 3...32 <i>e.g. FebSamples 32</i>
FebGains	1..3	0	int, 0=auto, 1=low, 2=med, 3=high Note that in data the notation is different (Section 3.1.1): 0=low, 1=med, 2=high <i>e.g. FebGains 1 2</i>
FebAddr	1..8	0	hex, SPAC addresses of installed FEBs, listed in the increasing order of the corresponding miniRODs <i>e.g. FebAddr 0x28 0x26 0x3F 0x22 0x30 0x3A 0x12 0x21</i>
FebTimeout	1	1000	int, miniROD time-out, in ms
FebDacOffset	1	0xc00	int, pedestal setting. Value 0xc00 corresponds to about 1000 ADC counts. Increasing this parameter results in <i>decreasing</i> of the pedestal. Currently, the default value is the unique setting for all the FEBs.
FebAutoGainThr	2	1150 3500	int, the thresholds (in terms of ADC counts in medium gain) for switching to the high and low gains, respectively. These parameters are only meaningful if the auto-gain mode is selected (FebGains 0)
FebFirstSample	1	3	int, the sample number (counting from 0) whose pulseheight a FEB has to use to choose the gain in the auto-gain mode. A cyclic shift is performed on the first (FebFirstSample+1) samples, as <a href="#">described</a> in Section 3.1.2  This parameter is ignored for the fixed gain mode(s), when it is reset to 0 (so no sample re-ordering occurs and all the samples appear in data in the chronological order).

<i>key</i>	<i># of values</i>	<i>Default value (s)</i>	<i>Value type, description and range</i>
<b>FebReadDelay</b>	1	0x11	int, the crude signal time tuning ("trigger latency"), n 25 ns steps
<b>TtcPdgDly</b>	1	200	int, the fine signal time tuning (common PDG delay), in 50 ps steps
<b>TtcFanDly</b>	8	0	int, individual signal time tuning for different FEBs, in 2.5 ns steps.

## 2.6 Keys related to CAMAC

All keywords starting with `Cam` refer to CAMAC readout. They are usually defined in the `par/cam/...par` file specified by the `ConfCam` key in the run header. There are two groups of such keys: the ones specifying slots (station numbers) of all CAMAC hardware modules used in the readout – *module locatorss*, and the ones describing a complete chain of CAMAC calls necessary to read specific detectors – *readout descriptors*.<sup>6</sup> An additional `CamSlTest` key is used for CAMAC debugging purposes. See Appendix 2-A for examples of `Cam...` keys.

Module locators have a variable number of integer values, as there can be several CAMAC modules of the same type (e.g., LeCroy 2249A ADC). The key values are the corresponding slot numbers (ranging from 1 to 23). The syntax permits the crate number to be specified (as  $C*100+N$ ), but if there is only one CAMAC crate, its number is assumed by default. The available module locator keywords (corresponding to “classes” internally defined in the DAQ) are listed in Table 1.

<code>CamBorer</code>	BORER dataway display
<code>Cam2228A</code>	LeCroy 2228A TDC
<code>Cam2249A</code>	LeCroy 2249A ADC
<code>CamSc2551</code>	SEN 2551 Scaler
<code>CamOR2088</code>	SEN 2088 Output Register
<code>CamPattB</code>	Pattern Unit B
<code>Cam4448</code>	LetCroy 4448 48-ch Coincidence
<code>CamRTC</code>	Real Time Clock

<sup>6</sup> Strictly speaking, readout descriptors alone would be sufficient for defining the CAMAC readout. However, the module locators introduce a useful redundancy, helping to reduce a probability of mistakes. Any N (slot number) appearing in readout descriptors is required to be described in module locators. This permits further checks: for example, the BPC time readout checks that the N corresponds to a 2228A TDC and the F is a valid function for it. DAQ has an internal library of “classes” for all CAMAC devices used in the readout.

The readout descriptor defines a list of CAMAC NFA's, encoded as integers  $N*10000+F*100+A$ , for each DAQ part read out or interfaced via CAMAC, see Table 2.

CamEvClock	Trigger timing wrt the 40 MHz clock
CamPattern	Trigger bit pattern, r/o bits
CamBpc_X	X=1-6; beam chambers
CamTime	Trigger signal timing, TDC
CamTail	Tail catcher, ADC
CamBeam	Beam detectors, ADC
CamOutReg	Software pulser

## 2.7 Keys referring to the configuration files

At a start of run, a global configuration file referred to by the RunConf key is specified via the Run Control Panel. This file itself contains references to other configuration files (via ConfTrig, ConfFeb, ConfCal, ConfCam keys)<sup>7</sup>. The file names are relative to the base directory defined by the \$DAQ\_CONF\_PATH environment variable, usually /raid/daq/config

Examples:

```
RunConf Special/HighG_beam_ped.v0
ConfTrig par/trig/0_300_0.par
ConfFeb par/fe/highG_7sam.par
ConfCal par/cal/cal0.par
ConfCam par/cam/test_cam.par
```

<sup>7</sup> As well as some other records which are not (yet) configurable via the Run Control Panel, like ReadOutMask, RunDebug etc).

## 3 Event sub-blocks

The event sub-blocks other than the event header are described below.

### 3.1 FEB Data

The information about the FEB data format is sparse and not always consistent. The main source is [Ref. \[3-3\]](#). A very detailed description of the FEB operation is in [Ref. \[3-1\]](#). A concise description of the miniROD operation is given in [Ref. \[3-2\]](#). miniRODs preserve the format of the FEB ADC data and control words. However, the order in which the FEB ADC data is read-out from the miniROD VME ports is defined by the miniROD hardware (the FEBs send the bit streams from all ADCs simultaneously, over the parallel bus).

A simple code to unpack one FEB and get a pointer to the next FEB is given in the [Appendix 3-A](#). A summary of FEB- and miniROD related run header key-records is given in [Appendix 3-B](#).

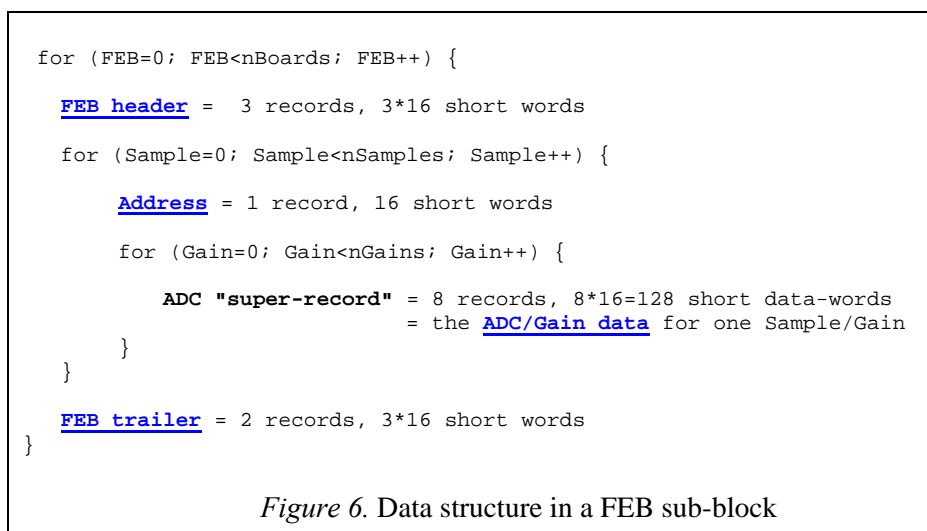
#### 3.1.1 The sub-block structure

```

SubBlockId   : 0xFF02
Len          : (nBoards*(3+nSamples*(1+nGains*8)+2)*16) short words

```

The data part is sequence of logical “records” of 16 short integer words. The contents of a record varies depending on the context within the sub-block, as illustrated by Figure 6: it can be one of FEB **header** or **trailer** records, an **address** record, or a **ADC/Gain data** record,



The parameters `nBoards`, `nSamples`, `nGains` come from the run header:

```

nBoards = number of values in the miniROD key of the run header
           (e.g., rh_get_int("miniROD", &nBoards, ...) )
nSamples = the value of the FebSamples key of the run header
           (e.g., rh_get_lint("miniROD", &nSamples) )
nGains = number of values in the FebGains key of the run header
           (e.g., rh_get_int("FebGains", &nGains, ...) )

```

The FEB control words (header and trailer) contain some fixed patterns (words), which can be checked to test the data integrity.

```

Feb Header:
  record 1: 16* 0xffff = "start-of-event" words
  record 2: ????.
           Example: 4303 4303 0302 0302 0301 0301 4300 4300
                   0307 0307 4306 4306 4305 4305 0304 0304
  record 3: 16* (bits 0-11: "bunch-crossing number", bit 14: OP)

```

The meaning of “record 2” remains unclear. My guess is: bits 0-3=ALTERA #, bits 8-9: ? (always 3), bit 14=OP. See [Ref \[3-3\] p.4](#) for more insight...

```

Feb Trailer:
  record 1: 16* ( bit 0: 1, bits 1-11: "error word", bit 14: OP)
           "error word" = 0 means "no errors"
  record 2: 16* 0x0000 = "end-of-event" words

```

The **Address** record should contain the Gray-coded SCA cell number (bits 0-7) and a certain number *nm* (bits 8-11), plus bit 14=OP. According to Ref.[3-3], *nm* is either the ADC number, or the Sample number. The former seems to be excluded, while the latter is yet to be checked. Quite a few tests are applicable here (Grey code, identity of cell numbers). A function `samplecell` to convert the coded cell number to a sequential number is available.

In the ADC data **super-records**, each word contains the **ADC/Gain data** from a certain FEB input channel. The correspondence between the FEB input channels and the word numbers in the super-record is described in the [Appendix 3-C](#) and, in more detail, in Ref [3-8].

Each ADC **data-word** is formatted as follows ([Ref.\[3-3\], p.3](#)):

```

unsigned int ADCvalue      : 12 bits; // bits 0-11
unsigned int Gain         : 2 bits; // bits 12-13 (3=high, 2=medium, 1=low)
unsigned int ParityOdd    : 1 bit ; // bit 14 (OP = odd parity)
unsigned int ADCflag=0    : 1 bit ; // bit 15

```

Bits 12-15 have special meaning in all FEB records:

- **bits 12 and 13** are both 0, except for ADC/Gain data and start-of-event words;
- **bit 15** is always 0, except for start-of-event words;
- **bit 14** is the odd parity (e.g., 1 if all other bits are 0), except for start-of-event and end-of-event words.



### 3.1.2 Special considerations for auto-gain runs

In the "auto-gain" mode, one has to determine the sample corresponding to the signal peak and program the Altera FPGA controller accordingly. The FCal test beam trigger timing had been adjusted to have the signal peak close to sample 3 (counting from sample 0), in all FEBs. The FCal DAQ conveys this information to the Altera via the `FebFirstSample` configuration key-record ([Section 2.5](#)).

It turned out that if a non-zero `FebFirstSample` parameter is loaded, the Altera performs a cyclic shift of all the samples up to the sample=`FebFirstSample`.<sup>8</sup> For example, if `FebFirstSample` is 3, then the sample order in the data is

```
3, 0, 1, 2, [4, 5, 6...]
```

Therefore, the offline programs should re-order the samples if (and only if, see the remark below) the parameter `FebGains` in the run header is set to 0:

```
FebGains 0 // 0=auto, 1=low, 2=med, 3=high
```

This and all other important parameters can be obtained from the run header ([Section 2.5](#)).

**Important remark:** For fixed-gain runs (non-zero `FebGains`) the parameter `FebFirstSample` must be ignored, even if it has a non-zero value in the run header (as was the case for the early FCal test beam runs, up to run 1435), because DAQ resets it to zero internally. The sample re-ordering according to `FebFirstSample` should be applied only if `FebGains` key is non-zero.

## 3.2 Beam chambers data

### 3.2.1 The sub-block structure

```
SubBlockId   : 0xFF05
Len          : nBPC*6 short words
```

The sub-block contains raw amplitude (ADC) and time (TDC) data from the beam profile chambers (BPCs). The data part of the sub-block, as shown in [Fig. 7](#), consists of *nBPC* groups of 6 integers, where *nBPC* comes from the run header:

```
nBPC = number of enabled BPCs = number of values in the Bpc key in the run header,
      (e.g., rh_get_int("Bpc", &nBpcs, ...) )
```

---

<sup>8</sup> That the samples can be sent in a non-sequential order was indicated in Refs.[3-3] and [3-4], as a "possibility". I could not find any other documentation explicitly describing this behaviour.

```

for (BPC=0; BPC<nBpc; BPC++) {
    word 0 : X-plane ADC      } 10 bits data + 1 overflow bit
    word 1 : Y-plane ADC      } (meaningful values: <1024)

    word 2 : TDC Xright } 11 bit data + 1 overflow bit
    word 3 : TDC Xleft  } (meaningful values: <2048)
    word 4 : TDC Yup     }
    word 5 : TDC Ydown   }
}

```

Figure 7. Data structure in a BPC sub-block

### 3.2.2 A brief description of the BPCs and how to interpret the BPC data

The technical details about the BPCs can be found in Ref.[3-5].

*BPC numbering:* the BPCs are numbered from 1 to 6, according to their physical location along the beam:

- BPC 1 and 2 are the most upstream chambers (new X-Y type);
- BPC 3 and 4 are the "chambers" located midway to the cryostat; they are, actually, 4 old single-projection ITEP chambers, arranged in two X-Y pairs;
- BPC 5 and 6 are the chambers installed in front of the cryostat (new X-Y type).

*Z-positions* (in mm, related to Ch.1-X) (courtesy V.Epstein):

Z(1X) = 0;      Z(1Y) = 31;      Z(2X) = 177; Z(2Y) = 208;  
Z(3X) = 11076; Z(3Y) = 11154; Z(4X)=11219; Z(4Y) =11294;  
Z(5X) = 27645; Z(5Y) = 27676; Z(6X)=27745; Z(6Y) =27776;

*Meaning of the data:* each chamber has two wire planes measuring horizontal (X) and vertical (Y) positions of a beam particle. One plane measurement consists of a pair of TDC readings (right/left for X or up/down for Y) and an ADC reading (cathode signal amplitude, needed for event selection and corrections).

The time measurement is done with 11-bit LeCroy 2228A TDCs (Ref.[3-6a]). Bits 0-10 of the TDC words are data, bit 11 signals overflow. Thus, meaningful TDC values are 0-2047. The amplitude measurements are done with LeCroy 2249A ADC (Ref.[3-6b]).

The beam position can be derived from the TDC measurements<sup>9</sup>:

$$X_i = C1_{xi} \cdot (Xleft_i - Xright_i - C2_{xi})$$

$$Y_i = C1_{yi} \cdot (Ydown_i - Yup_i - C2_{yi})$$

The resulting values are in *mm*, with +ve X pointing to Geneva, -ve X – to Jura, +ve Y pointing up and -ve Y – down. This formula is accurate down to better than 0.5 *mm*. The corresponding calibration constants C1 and C2 are listed in Table 3. To obtain the ultimate accuracy of better than 150  $\mu$  per plane, one has to apply a special calibration, on the run-by-run basis (work in progress).

<sup>9</sup> This is a simplified formula, good for the cases when the an accuracy of about 1 mm is sufficient. See Ref. [3-5] for more details.

Up to run 1722		as of run 1740		Up to run 1722		as of run 1740	
C1x(1)=	0.0497	0.0518		C1y(1)=	0.0496	0.0523	
C1x(2)=	0.0493	0.0513		C1y(2)=	0.0506	0.0521	
C1x(3)=	0.0533	0.0532		C1y(3)=	0.0527	0.0525	
C1x(4)=	0.0527	0.0526		C1y(4)=	0.0539	0.0539	
C1x(5)=	0.0488	0.0489		C1y(5)=	0.0498	0.0499	
C1x(6)=	0.0476	0.0477		C1y(6)=	0.0480	0.0480	
C2x(1)=	16.	34		C2y(1)=	-4.	2.	
C2x(2)=	-9.	2.		C2y(2)=	-11.	-13.	
C2x(3)=	-26.	-27.		C2y(3)=	-4.	-3.	
C2x(4)=	-7.	-9.		C2y(4)=	3.	5.	
C2x(5)=	-13.	-12.		C2y(5)=	-42.	-42.	
C2x(6)=	-24.	-22.		C2y(6)=	-20.	-19.	

### 3.2.3 BPC-related Run Header keys

*Obligatory:* CAMAC read-out descriptors (NFAs), see Section 2.6.

```
CamBpc_x <list of NFAs> // x=1-6
```

Example:

```
Cam2228A 2 4 8 10 // TDC module locator
Cam2249A 12 14 18 // ADC module locator
CamBpc_1 100000 100001 20000:20003 // adc X,Y tdc R,L,U,D
...
CamBpc_6 100010 100011 40004:40007
```

*Optional:* selection of BPC's to read (default: all); usually defined in the par/run/...par file.

```
Bpc <list of enabled BPCs>
```

Example:

```
Bpc 1 2 5 6 // the BPCs will be read-out in that order!
```

### 3.3 Beam detectors, Tail Catcher, TIME data

#### 3.3.1 Beam detectors sub-block

**SubBlockId** : 0xFF06  
**Len** : 11 short words

<i>Word</i>	<i>meaning</i>	<i>ch</i>	<i>CAMAC module</i>
0	S1	0	ADC 2249A (N12 "BEAM")
1	S2	1	
2	S3	2	
3	Veto	3	
4	SFiber	4	
5	SFiber-amp	5	amplified??
6	Muon	word 6	
7	S2	0	TDC 2228A (N2 "TIME/BEAM")
8	S3	1	
9	Veto OR	2	
10	TC "antenna"	7	ADC 2249A (N12 "BEAM") added on June,27 2003

#### 3.3.2 Tail Catcher sub-block

**SubBlockId** : 0xFF04  
**Len** : 12 short words

<i>Word</i>	<i>meaning</i>	<i>ch</i>	<i>CAMAC module</i>
0-5	TC1-TC6 High gain	0-5	ADC 2249A (N14 "TAIL")
6-11	TC1-TC6 Low gain	6-11	

#### 3.3.3 Time sub-block: id 0xFF03

**SubBlockId** : 0xFF03  
**Len** : 2 short words

<i>Word</i>	<i>meaning</i>	<i>ch</i>	<i>CAMAC module</i>
0	40 MHz clock	3	TDC 2228A (N2 "TIME/BEAM")
1	40 MHz clock+12.5ns	4	
2	scaler		time elapsed since the previous particle crossing
3	scaler		same, downscaled

This sub-block is used to determine the phase of the calorimeter signal with respect to the 40 MHz TTC (sampling) clock. The TDC is started by the trigger; the ch. 3 measures the TTC Clock signal (that is, the falling edge of the Clock nearest to the trigger) and the ch. 4 measures the same signal delayed by  $\sim 10$  ns<sup>10</sup>.

<sup>10</sup> The TTC Clock is a 40.08 MHz pulse generator, with a pulse width somewhat smaller than a half-period. The delayed Clock is obtained by inverting the Clock signal.

The redundancy permits a) to calibrate the TDC and b) to resolve the ambiguities occurring when the trigger and the Clock pulse arrive almost simultaneously to the TDC.

### 3.3.4 Notes about the CAMAC modules

- ADC 2249A (N12 "BEAM"), 10 bits (0-1023)
- TDC 2228A (N2 "TIME/BEAM"), 50 ps res, range: 11 bits (0-2023, bit 11=overflow); Start: S1\*Gate
- TDC 2228A (N2 "TIME/BEAM"), 50 ps res, range: 11 bits (0-2023, bit 11=overflow); Start: S1\*Gate

## 3.4 Calibration Board Stamp

### 3.4.1 The sub-block structure

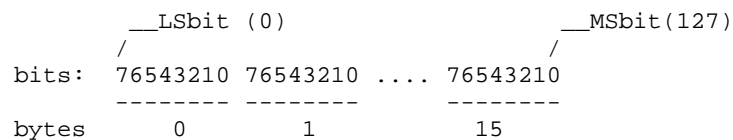
```
SubBlockId : 0xFFFF
Len        : 11 short words (22 bytes)
```

The sub-block contains a complete pulser board information (a “stamp”) for a given event. The data is a direct copy of the byte string read from the pulser board after it had been prepared to deliver the calibration pulse for the given event. This sub-block appears only in pulser events (type 2).

**Data format:** a string of 22 bytes

```
bytes 0-15 (16 bytes, 128 bits): the bit-pattern of the pulsed channels
                                (see next subsection for further details)
bytes 16-19 (4 bytes, 32 bits) : the DAC value (pulse amplitude)
                                byte 16 = LSByte
                                ...
                                byte 19 = MSByte
                                For example: f8 2a 00 00 means DAC=11000

byte 20 (1 byte, 8 bits) : the delay value, in units of ~1 ns
byte 21 (1 byte, 8 bits) : error word (OK=0, if non-zero, the
                                event should be discarded)
```



*Figure 8. Structure of the channel pattern in the CalStamp sub-block*

### 3.4.2 How to decode the channel pattern

1. Considering bytes 0-15 as a contiguous bit-string (Fig. 8), make a list of non-zero bits, with the bits numbered from 0 to 127.
2. For bit numbers in the ranges (32-63) and (96-127), the bit number coincides with the FEB input channel number. For all other bits, the odd and even numbers must be swapped (e.g., bits 0 and 1 correspond to the channels 1 and 0, respectively; bits 64 and 65 - to channels 65 and 64 etc).

The code in Fig. 9 illustrates the channel decoding (/home/daq/Daq/include/my\_event.h and /home/daq/Daq/lib/my\_event.c)

```

typedef struct {
    union{
        struct {
            unsigned char pattern[16];
            unsigned char dac[4];
            unsigned char delay;
            unsigned char error;
        };
        unsigned char data[22];
    };
} Cal_Stamp_t;
....
//-----
void cal_stamp_dump (Cal_Stamp_t *cal_stamp) {
//-----
// Print-out the calibration stamp
int nch,n,j,k;
unsigned long *Dac;
unsigned char ch;

printf("Cal_Stamp: "); for(j=0;j<22;j++) printf("
%02x",cal_stamp->data[j]);

// decode the channel pattern (convert the pattern bits numbers
into FEB pin numbers)
//
n=0;
for(j=0;j<16;j++) {
ch=cal_stamp->pattern[j];
for (k=7;k>=0;k--) {
if((ch>>k)&1) {
if( (n/32)%2 )
nch=n;
else
nch=n%2?n-1:n+1;
printf("n=%d nch=%d",n,nch);
}
}
n++;
}
}
Dac = (unsigned long *)cal_stamp->dac;
printf("\n ch=%d DAC=%d delay=%d error=0x%02x \n",
nch,*Dac,cal_stamp->delay,cal_stamp->error);
}

```

*Figure 9. An example of the channel pattern decoding in the Calibration Board Stamp sub-block*

## Appendix 2-A: an example of a complete run header block

```

* -----µ/raid/daq/config/par/runs/run1461.par
RunNumber 1461
RunType 0
RunConf Special/HighG_beam_ped.v0
DataStore 1 /raid/data
BeamMomentum 200 GeV/c
BeamParticle 11 // e-
BeamSpot 5 // 4D
MaxEvents 1000
* ----- /raid/daq/config/Special/HighG_beam_ped.v0

ConfTrig par/trig/0_300_0.par
ConfFeb par/fe/highG_7sam.par
ConfCal par/cal/cal0.par
ConfCam par/cam/test_cam.par
ReadOutMask Fcal Bpc Time Tail Beam

miniROD 1:8

Bpc 1:6

* ----- /raid/daq/config/par/trig/0_300_0.par
TrigBeam 0
TrigPed 300
TrigFcal 0
* ----- /raid/daq/config/par/fe/highG_7sam.par
FebSamples 7
FebGains 3
FebAddr 0x28 0x26 0x3F 0x22 0x30 0x3a 0x21 0x12

miniROD 1 2 3 4 5 6 7 8

FebTimeout 1000
FebDacOffset 0xc00
FebAutoGainThr 1350 3500
FebReadDelay 0x11
FebFirstSample 3
TtcPdgDly 150
TtcFanDly 0 0 0 0 0 1 0

* ----- /raid/daq/config/par/cam/test_cam.par
CamBorer 1
Cam2228A 2 4 8 10
Cam2249A 12 14 18
CamSc2551 20
CamOR2088 22
CamPattB 15
Cam4448 6
CamRTC 23
CamEvClock 230000 230001
CamPattern 60200 150200
CamBpc_1 180000 180001 40000:40003
CamBpc_2 180002 180003 40004:40007
CamBpc_3 180004 180005 80000:80003
CamBpc_4 180006 180007 80004:80007
CamBpc_5 180008 180009 100000:100003
CamBpc_6 180010 180011 100004:100007
CamTime 20003:20004
CamTail 140000:140011
CamBeam 120000:120006 20000:20002
CamOutReg 221700
CamSltest 200

RunDate 20030620
RunTime 155336

```

## Appendix 2-B: rhlib package, to handle the run-header records

**Author:** P. Gorbunov

**Source:** /home/daq/daq/include/rhlib.h and /home/daq/daq/lib/rhlib.c

**Purpose:** The internal DB to manage the run header data

These codes are not DAQ-specific and can be used in offline applications (including Fortran). The following C-functions are available:

**void rh\_init(void);**

To reset the run header structure. The existing structure is dropped.

**void rh\_import(void \*rh\_addr, int rh\_len) ;**

To drop the existing run header structure and copy a new structure of rh\_len bytes from the pointer rh\_addr

**void rh\_export(void \*\*rh, size\_t \*rh\_len) ;**

Returns the address (rh\_addr), the total size in bytes (rh\_len) of the run header structure.

**void rh\_put\_str( char \*key, char \*str) ;**

To append the keyword "key" + string "str" to the run header.

**int rh\_get\_str(char \*key, int \*n, char \*\*str, int max\_len, int max\_val) ;**

To get string fields from the "key" and store them in the array of pointers "str" of size "max\_val", each pointing to a string of at least "max\_len" bytes.

**void rh\_put\_int(char \*key, int n\_val, int \*values) ;**

To add the key-record "key" with "n\_val" integer values.

**int rh\_get\_int(char \*key, int \*n, int \*val) ;**

To retrieve int value(s) with the key-record "key". *Range* fields f:l[;s] ( l>=f, s>0 ), as well as *multiplier* fields v\*m (m>0) are interpreted.

**int rh\_get\_lint(char \*key, int \*value) ;**

To get a single int value from the "key".

**void rh\_put\_lint(char \*key, int value) ;**

To add a key-record with a single integer value.

**void rh\_read (char \*header\_fn) ;**

To append the contents of file "header\_fn" to the run header structure.

**void rh\_write (char \*header\_fn) ;**

To write the run header in ASCII form to file header\_fn.

**void rh\_dump (void);**

To dump the contents of the run header structure

**int rh\_find\_first(char \*key) ;**

Returns the record number for the first appearance of "key", or -1 if "key" is not found

**int rh\_find\_last(char \*key) ;**

Returns the record number for the last appearance of "key", or -1 if "key" is not found



```
int rh_give(char *keyp, char *valp, int *irc, int keyl, int vall) ;
```

Iterates over the run header records. If `*irc==0`, starts from the beginning. Returns the keyword in `keyp` (up to `keyl` characters), and the value field string in `valp` (up to `vall` characters). Returns `*irc=-1` if the run header is exhausted, otherwise `*irc` is the recird length.

Fortran-callable versions of the following routines are available: `rh_init`, `rh_read`, `rh_dump`, `rh_get_int`, `rh_put_int`, `rh_get_lint`, `rh_give`, `rh_write`.

The source codes can be found in `/afs/cern.ch/user/p/petr/public/fcaltb/lib` .

## Appendix 3-A: An example of the FEB-data unpacking code

<http://atlas-fcaltb.web.cern.ch/atlas-fcaltb/Memos/DAQ/Appendix3A.c>

```

#include "rplib.h" // pcfcal02:/home/daq/include
#include "my_event.h"

short *miniROD_unpack_example(short *FEBdata, rdFEB_t *Event ) {

// an example of FEB sub-block unpacking
// 19/06/03 PG: sample-reordering
// 15/06/05 PG: direct channel mapping
//
int i,j,k,next,nch;
int ns=run_par.nsamples, // nb. of samples, from the run header
    ng=run_par.ngains; // nb. of gains, from the run header

int sort[128]={
63, 55, 47, 39, 31, 23, 15, 7, 127, 119, 111, 103, 95, 87, 79, 71,
62, 54, 46, 38, 30, 22, 14, 6, 126, 118, 110, 102, 94, 86, 78, 70,
61, 53, 45, 37, 29, 21, 13, 5, 125, 117, 109, 101, 93, 85, 77, 69,
60, 52, 44, 36, 28, 20, 12, 4, 124, 116, 108, 100, 92, 84, 76, 68,
59, 51, 43, 35, 27, 19, 11, 3, 123, 115, 107, 99, 91, 83, 75, 67,
58, 50, 42, 34, 26, 18, 10, 2, 122, 114, 106, 98, 90, 82, 74, 66,
57, 49, 41, 33, 25, 17, 9, 1, 121, 113, 105, 97, 89, 81, 73, 65,
56, 48, 40, 32, 24, 16, 8, 0, 120, 112, 104, 96, 88, 80, 72, 64
};

const int Nswrec=16; // short words per FEB data "record" (Section 3.1.1)

short *p= FEBdata;
int gain; // the gain
int s; // the sample number
short a; // ADC data word

static int sort_samples[32]=
{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,
 11,12,13,14,15,16,17,18,19,20,
 21,22,23,24,25,26,27,28,29,30,31};

// re-order the samples for auto-gain runs
//
if( (run_par.the_gains[0]==0) && // check the gain-selection mode
    (sort_samples[0]=run_par.first_sample)!=0) { // check for the FebFirstSample != 0
    for(j=0;j<run_par.first_sample;j++) sort_samples[j+1]=j;
}

p += Nswrec*3; // skip the FEB header
for(i=0; i<ns; i++) { // loop over the samples

    s=sort_samples[i];

    p += Nswrec; // skip the cell record
    for (j=0; j<ng; j++) { // loop over the gains
        for( next=0; next<128; next++ ) { // loop over the channels
            nch=sort[next];
            a=*p;
            gain= ((a>>12)&3) - 1;
            Event->adc[nch ][gain][s] = a& 0xffff;
            p++;
        }
    }
}

// skip the FEB trailer and return the pointer to the next FEB
return (p += Nswrec*2);
}

```

## Appendix 3-B FEB- and miniROD-related run header keys

*Disclaimer:* the numbers are not representative...

```

FebSamples      7
FebGains        0 // 0=auto, 1=low (0 in data), 2=med (2 in data), 3=high (2 in
data)
*
FebAddr         0x28 0x26 0x3F 0x22 0x30 0x3a 0x21 0x12 // FEB addr
miniROD         1  2  3  4  5  6  7  8 // the miniRODs to be
read-out
FebTimeout      1000 // ns, miniROD time-out setting
FebDacOffset    0xc00 // DACoffset = coupleDACoffsets. Offset up, ped down.
0xc00: ped ~ 1000
FebAutoGainThr  1350 3500 // auto-gain thresholds
FebReadDelay    0x11 // delay up, peak ->; delay down, peak <-
* -----
FebFirstSample  3 // the peak sample number (starting from 0)
* -----
*
* TtcPdcDly     0 // Calib pulse delay, 0-4095, in steps of 25 ns (max=6.5 us)
TtcPdgDly      150 // PDG delay 0-4095, in steps of 50 ps (max=200 ns) delay up,
peak <-
TtcFanDly      0 0 0 0 0 0 1 0 // FEB Fan-out delays, 0-7, in steps of 2.5 ns (max =
20 ns)

```

## Appendix 3-C A structure of the FEB ADC “super-record”

A FEB ADC **super-record** (Section 3.1.1) is a sequence of 128 short integers containing a complete FEB ADC data for one sample and one gain. Each word contains packed **ADC** and **Gain** values for one FEB channel. The correspondence between the FEB channels and the word numbers is shown in Table A3C.

Logically, the read-out order represents repetitive loops over all 16 ADCs of the FEB:

```

loop over ADC channels (7,6,..,0) {
    for each channel, read 16 ADCs in this order: (7,6,..0), (15,14,..8)
}

```

Each ADC has 8 channels and accepts signals from 8 consecutive input pins connected to a pair of 4-channel pre-amplifier hybrid circuits located on both sides of the FEB. The printed conductor path (track) lengths for these 8 channels are roughly the same (within 15%), but the average track length varies from ~80 mm to ~24 mm for different ADCs. The two halves of a FEB (ADC 0-7 and ADC 8-15) have similar (though not identical) layouts of the input conductors, namely is ADC 15 is similar to ADC7, ... ADC 8 is similar to ADC 0. **Thus, a super-record consists of consecutive groups of 8 words, each group reflecting 8 basic layouts of the input analog part of the FEB PCB.**

**Table A3C:** The correspondence between the FEB channels and the super-record word numbers. All items are numbered starting from 0.

<http://atlas-fcaltb.web.cern.ch/atlas-fcaltb/Memos/DAQ/Appendix3C.c>

```
// FEBchannel [ word_number ]
int FEBchannel [128]={
  63, 55, 47, 39, 31, 23, 15, 7, 127, 119, 111, 103, 95, 87, 79, 71,
  62, 54, 46, 38, 30, 22, 14, 6, 126, 118, 110, 102, 94, 86, 78, 70,
  61, 53, 45, 37, 29, 21, 13, 5, 125, 117, 109, 101, 93, 85, 77, 69,
  60, 52, 44, 36, 28, 20, 12, 4, 124, 116, 108, 100, 92, 84, 76, 68,
  59, 51, 43, 35, 27, 19, 11, 3, 123, 115, 107, 99, 91, 83, 75, 67,
  58, 50, 42, 34, 26, 18, 10, 2, 122, 114, 106, 98, 90, 82, 74, 66,
  57, 49, 41, 33, 25, 17, 9, 1, 121, 113, 105, 97, 89, 81, 73, 65,
  56, 48, 40, 32, 24, 16, 8, 0, 120, 112, 104, 96, 88, 80, 72, 64};

// word_number [ FEBchannel ]
int word_number [128] = {
  119, 103, 87, 71, 55, 39, 23, 7, 118, 102, 86, 70, 54, 38, 22, 6,
  117, 101, 85, 69, 53, 37, 21, 5, 116, 100, 84, 68, 52, 36, 20, 4,
  115, 99, 83, 67, 51, 35, 19, 3, 114, 98, 82, 66, 50, 34, 18, 2,
  113, 97, 81, 65, 49, 33, 17, 1, 112, 96, 80, 64, 48, 32, 16, 0,
  127, 111, 95, 79, 63, 47, 31, 15, 126, 110, 94, 78, 62, 46, 30, 14,
  125, 109, 93, 77, 61, 45, 29, 13, 124, 108, 92, 76, 60, 44, 28, 12,
  123, 107, 91, 75, 59, 43, 27, 11, 122, 106, 90, 74, 58, 42, 26, 10,
  121, 105, 89, 73, 57, 41, 25, 9, 120, 104, 88, 72, 56, 40, 24, 8};
```

## References

### Chapter 1:

[1-1] P. Loch, .....doc, .... April 2003, the original event format specification.

### Chapter 3:

- [3-1] ATL-AL-EN-0009 *Design of the ATLAS LAr Front End Board*,  
<http://www.nevis.columbia.edu/~atlas/electronics/Module0FEB/febdocument.ps> and  
<http://cern.ch/atlas-fcaltb/Memos/Hardware/FEB0/ATL-AL-EN-0009.ps>
- [3-2] LARG-ELEC-3, *miniROD board Draft Specifications*, and references therein.  
</afs/cern.ch/user/p/petr/public/perrodo/NOTES/testbeam-mra/mradoc.ps>
- [3-3] ATL-AL-LAL-ES-1.0 *Format for the Data read out from the front-end boards*, E. Auge et al, 1997,  
<http://cern.ch/atlas-fcaltb/Memos/Hardware/FEB0/ATL-AL-LAL-ES-1.0.ps>
- [3-4] P.Loch, F.Lanni: private communications
- [3-5] ITEP BPC Note1, 14-Sep-2003, *ITEP beam chambers*,  
<http://cern.ch/atlas-fcaltb/Memos/Hardware/BPC/>
- [3-6] a) <http://www-esd.fnal.gov/esd/catalog/main/lcrynim/2228a-spec.htm>  
 b) <http://www.fnal.gov/projects/ckm/jlab/2249a-spec.htm>
- [3-7] A.Hincks, 20-Aug-2003, *Reconstructing the Trigger Delay from the TTC Values*,  
[http://cern.ch/atlas-fcaltb/Memos/Analysis/Adam\\_Hinks/timing.ps](http://cern.ch/atlas-fcaltb/Memos/Analysis/Adam_Hinks/timing.ps)
- [3-7] P.Gorbounov, 15-June-2005, *Channel mapping in FCal TB 2003 Data*,  
<http://cern.ch/atlas-fcaltb/Memos/DAQ/ChannelMapping2003.pdf>