Contents

GRETINA and auxilary data processing

torben lauritsen and I-Yang Lee

September 11, 2012

1 Goal of this document

After giving an introduction to the principles of gamma ray tracking, this document spells out a scheme for how to handle 'external' data with the GRETINA (GT) device. It is build on what we (at least some of us) have already done for GT+BGS, but documents the strategy and points out how it can be applied to GT+S800 – or any other future devices coupled to GT.

2 Tracking Intoduction

GRETINA has 28 Ge crystals, each with 36 segments, to cover $\frac{1}{4}$ of the 4π solid angle. The gamma ray tracking technique uses detailed pulse shape information from each of the segments. These pulses are digitized and analyzed in the digitizer modules which output the energy, time and the pulse shape. These raw data are called mode 3 data (the details of data format are in the following chapters). The pulse shapes are then used to determine the three-dimensional positions of all gamma-ray interactions, in a process called signal decomposition which outputs energy, time and (x,y,z) position of each interaction points. These data are called Mode 2 data. This information is then utilized together with possible auxiliary detector (i.e. BGS, or S800 etc.) data, and using the characteristics of Compton scattering, to track the scattering sequences of the gamma rays. The results of tracking which contain both Mode 2 data and the sequence of scattering, is called Mode 1 data. Tracking will give higher efficiency, better peak-to-total ratio and much higher position resolution then treating each crystal as individual detector. Particularly, the position of the first interaction point is important for the accurate Doppler correction of the gamma ray. This section will give a short discussion of the principle of the tracking program which converts Mode 2 to Mode 1 data. This program can be run online during the experiment and/or offline after the experiment for further refinement of the tracking parameters.

The tracking algorithm has three steps, clustering, tracking, and recovery. Clustering algorithm identifies interaction points which are potentially belonging to a gamma ray. This is needed to handle multiple gamma rays events. In this step, interaction points within a given angular separation as viewed from the gamma-ray source are grouped into a cluster. This clustering angle is called alpha which typically ranges from 10 to 25 degree depending on the gamma ray multiplicity of the events. For lower multiplicity events a larger alpha will includes all points belonging to a gamma ray, for higher multiplicity events a smaller alpha is preferred to avoid false clustering of several real gamma rays.

In the second step each cluster is evaluated by Compton tracking to determine whether it contains all interaction points belonging to a single gamma-ray and if it does the scattering sequence. Clusters with acceptable tracking figure-of-merit are kept. The tracking assumed the gamma ray deposit its full energy (i.e. the sum of the energy of the interaction points in a cluster is the energy of the incoming gamma ray). The full tracking algorithm tests all N! possible permutations (scattering sequences) of a cluster with N interaction points. Starting at the assumed first interaction point, the scattering angle θ_C can be calculated from

$$\theta_C = 1 + \frac{0.511}{E_\gamma} - \frac{0.511}{E'_\gamma} \tag{1}$$

With E_γ the energy of the incident gamma ray, $E'_\gamma = E_\gamma - E_1$ and E_1 the energy of the first interaction. The scattering angle can also be calculated from the position of the source, the first and second interaction points. If the two values of the scattering angle agree, it indicates that the assumed sequence source-point1-point2 is the correct sequence satisfying the Compton scattering energy-angle relation. In an actual detector due to finite energy and position resolutions, this agreement will not be exact and a figure-of-merit (FOM) is used to determine the acceptance of the scattering sequence.

$$\chi^2 = \frac{1}{N-1} \sum_{i=1}^{N-1} \left(\frac{\theta^i - \theta^i_C}{\sigma^i_\theta} \right)^2 \tag{2}$$

This FOM sums over all scattering points and gives an overall evaluation of a sequence. The sequence with the lowest FOM which is less than a limit is accepted. The FOM cut provide a trade-off between the efficiency and the peak-to-total ratio (P/T). A lower cut gives lower efficiency but a better P/T. The optimal value will dependent on the requirement of each experiment. This trade-off is also dependent on the number of interactions in a cluster, thus adjusting this parameter for each hit number event group is recommended.

Events with single interaction point cannot be tracked, for high-energy gamma rays, such as the 1332 keV line from 60Co, the single hit probability constitutes a negligible fraction of photopeak events. Thus, for gamma rays with E > 1 MeV, ignoring single hit events has a negligible effect on the efficiency while improving P/T because many of the low-energy single-hit events correspond to Compton scattering (where the Compton scattered gamma-ray left the detector). For lower energy incident gamma rays the fraction of single hit photopeak events increases and ignoring them will have a larger impact on the efficiency. However, if they are included, the peak-to-total ratio will be reduced. A special algorithm was developed to handle single hit events which accept or rejected single hit events based on the deposited energy and depth of the interaction. Lower energy events were accepted at shallower distances, and this distance was increased as the deposited energy increases. For energies above a certain value no single hit events were accepted. Use of this algorithm recovered 80-90% of the efficiency at lower energies (e.g., 200 keV) while improving the peak-to-total by a factor of 2 for the single hit events.

The clusters which failed the first attempt of tracking are further evaluated in the third step. Here, one tries to recover some of the remaining clusters by adding and subtracting individual interactions or clusters to and from other clusters. Monte-Carlo simulations indicate that given realistic assumptions on position and energy resolution, this step could further improve the efficiency and P/T. However, more tests are needed to determine the optimal parameters of this step.

3 Some basic assumptions for the rest of the document

- We work on mode 2 data (I.e., we operate on decomposed data)[1]
- The data from the 'external' detector may or may not be preprocessed
- We merge the external data into the data stream[2]
- We merge the data before tracking (i.e, we merge it to the mode 2 GT data)[3]

4 Create merged data

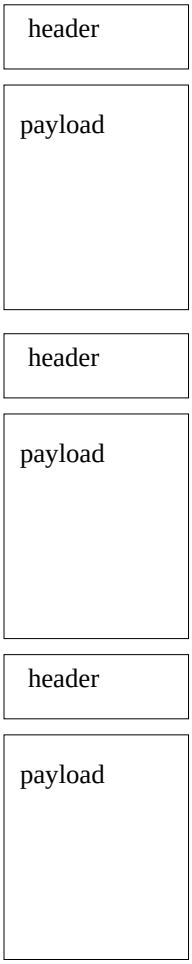

Figure 1: Merged mode 2 GT data

[1]For many applications we do need the tracking in order to determine the first interaction point and facilitate the best possible Doppler correction. This will be especially important when GT is operating at MSU.

[2]Another scheme was proposed which did not require merging of GT and external data; but it would have reduced the ability to plot external data vs. GT in the final data analysis, so it was not adopted.

[3]We could equally well have merged the data after tracking! Merging before the tracking has the advantage that we may be able to reduce the amount of GT data we need to perform tracking on. Tracking can be a very CPU intensive task – depending on what tracking options are switched on.

The merged mode 2 GT data stream has the general format shown in fig 1; simply a string of headers and data payloads. There is no file size limit. The header is a structure and looks like this:

```
struct gebData {
  int type;
  int length;
  long long timestamp;
};
```

where the type indicates what kind of data that follows in the 'payload' as well as the length of payload data (in bytes). There is a time stamp in the header as well, which *allows for time ordering and constructing coincidence data without knowing how to interpret the data payload itself!*.

The types we have defined now are

```
#define GEB_TYPE_DECOMP         1
#define GEB_TYPE_RAW            2
#define GEB_TYPE_TRACK          3
#define GEB_TYPE_BGS            4
#define GEB_TYPE_S800_RAW       5
#define GEB_TYPE_NSCLnonevent   6
#define GEB_TYPE_GT_SCALER      7
#define GEB_TYPE_GT_MOD29       8
#define GEB_TYPE_S800PHYSDATA   9
```

There will be new types defined as new types of external data is encountered. New types of GT dat might be defined as well [4].

At LBL, Chris was the point man for generating a data stream that had _DECOMP and _BGS data in it (is the program for this task available online?). At MSU Dirk has created the data with the GEB_TYPE_S800PHYSDATA format which allows us non-S800 experts to apply the proper doppler correction making use of the information from the S800 spectrometer. The preprocessing greatly reduce the somewhat complicated S800 data to something simple that any physicist can understand . The exact format is defined on the MSU wiki [5].

Assumption: *the time stamps in the headers of the merged mode 2 data are already ordered.*

5 First task: Make coincidences

The merged mode 2 data is just a long string of data events with time stamps from both GT and the external device. Before we can go on, we must make coincidence events by inspecting the time stamps. The 'datacomb3' program (available from HTTP://www.phy.anl.gov/gretina) is an example of a program that does that. The output from 'datacomb3' simply looks as

[4] **If any data format changes, it should be assigned a new GEB_TYPE type!**

[5] The BGS data was actually not reduced (true?) and therefore not all that easy to process by physicists not familiar with the BGS device.

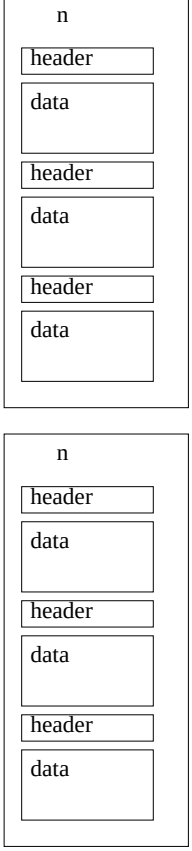

Figure 2: Mode 2 Coincidence data

shown in fig 2. For each coincidence event, the 'n' value tells how many header/payloads follow. These 'n' header/payloads are in coincidence within a certain time window specified in datacomb3 program (or whatever program you use).

Usually we do some data reductions at this point as well. E.g., we may require that there is one and only one external header/payload and at least one GT header/payload before we write the coincidence event out. We may also reject events where the energy from GT or the external devise is below some reasonable threshold.

6 Next: Tracking the GT data

To move from mode 2 to mode 1 data, *i.e., perform the tracking*, we run the data from the previous step through the trackMain program (avaliable from http://www.phy.anl.gov/gretina). This program will act only on the GT event types in the data stream and add to it the tracking result after the GRETINA mode 2 data. The data now may look as in fig 2. In this example the first coincidence data element is the external data, the next three are GT data elements. The input mode 2 data is repeated, and the tracked data is attached at the end. Again, any data type not of the GEB_TYPE_DECOMP type is just passed through without being touched by trackMain.

The tracking code may make use of the external data to determine where the gamma rays in the coincidence event were emitted from **if** this information can be extracted from the the

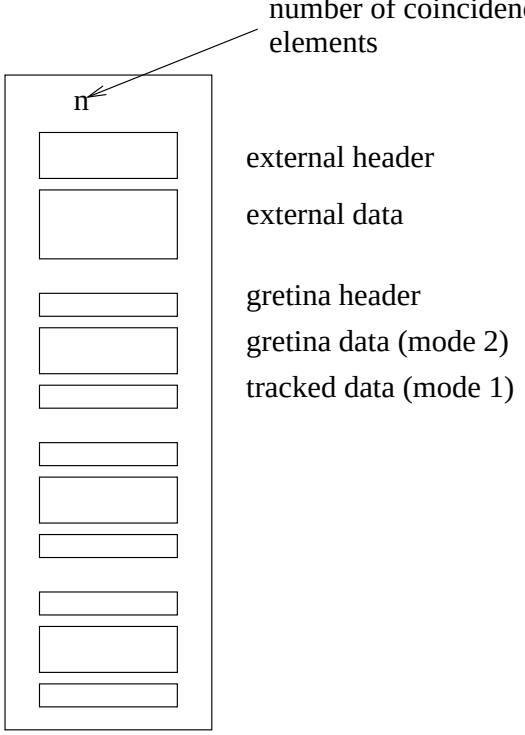

Figure 3: Tracked Coincidence data. In this example, the first event is external data and the following three events are GT data. The GT data is mode 1 data, so it has a gebData header, the repeated mode 2 data as well as the tracked (mode 1) data indicated by the boxes.

external data[6].

The data output from the tracking is the mode 1 data that we envisioned that GRETINA would provide the user with. However, to preserve the traces in the first experiments, we have typically accumulated mode 3 data on disk and generated the mode 1 data, as just described, as an off-line task.

7 Finally: Data sorting

The data output from the tracking code (trakMain) above can now be sent to your sorting code, where the real physics data analysis is performed. An example of such a sorting code is the **ctkana** program (available from http://www.phy.anl.gov/gretina) which sorts data into root files so they can be displayed and manipulated by with ROOT package from CERN (http://root.cern.ch). ctkana, or rather it's utilities, allow spectra and matrices to be written out in Radford formats.

ctkana sorts both the mode 1 data, gated on external (BGS) data and the original GT mode 2 data. combdata2 has a rather generous time window for making coincidences. The ctkana is able to set a more narrow time gates as well as generate background data.

ctkana is a simple minded program at the moment and needs to be expanded a great deal

[6]Code would have to be added to the tracking code to interpret the external data and extract this information. At the moment, the tracking knows nothing about what is in the external data payload.

for the final data analsys, if ctkana is to be used for this purpose. It will also have to be provided with code that can understand the S800 preprocessed data when gretina moves to MSU if ctkana should be used there.

It is not the aim of this document to specify how to sort the tracked data, but more to specify the format of the mode 1 data to be sorted.

8 Using the zlib

For use with the DigitalGammaSphere (DGS) project, we have implemented the use of zipped files using the zlib. The data reduction id about a factor of two. If there is interst in it, we can implement the zlib compression for the GRETINA mode 1 data as well.

A The GT mode 2 data format

The gebData structure was defined on page 5 and the types we have at the moment are listed on page 5. If we further define these structures and types

```
typedef struct DCR_INTPTS {
    float x, y, z, e;
    int seg;                /* segment hit */
    float seg_ener;         /* energy of hit segment */
    } DCR_INTPTS;

struct crys_intpts {
  int type;
  int crystal_id;
  int num;
  float tot_e;
  long long int timestamp;
  long long trig_time;
  float t0;
  float cfd;
  float chisq;
  float norm_chisq;
  float baseline;
  int pad;     /* to ensure 8-byte alignment of struct */
  DCR_INTPTS intpts[MAX_INTPTS];
};

typedef struct crys_intpts CRYS_INTPTS;
typedef struct gebData GEBDATA;
```

then the tracked data has his format

```
typedef struct TRACK_STRUCT {
```

```
  int m;
  GEBDATA *gd;
  CRYS_INTPTS *inp;
} TRACK_STRUCT;
```

Here 'm' refers to the number of interactions points, defined in DCR_INTPTS, the decomposition task found and passed on for tracking. The (x,y,z,e) values passed in the DCR_INTPTS structure are the positions and energies the decomposition task found in the crystal (with ID crystal_id). The coordinates are the crystal coordinates and not world coordinates. The header contains other information or various kinds, which are not all used, or even filled, at the moment.

you may read the data something like this

```
  siz = read (inData, (char *) &track->n, sizeof (int));
  ptgd = track->gd;
  ptinp = track->inp;
  for (i = 0; i < track->n; i++)
    {
      siz = read (inData, (char *) ptgd, sizeof (GEBDATA));
      siz = read (inData, (char *) ptinp, ptgd->length);
      ptgd++;
      ptinp++;
    };
```

B The tracked data format

As already mentioned on page 6, the output from the tracking (mode 1 data) repeats the input (mode 2 data) and adds the tracked data (the 'cluster' array) as this

```
i1 = 0;
for (iCluster = 0; iCluster < *nClusters; iCluster++)
  if (Clstr[iCluster].valid)
    i1++;
siz = write (Pars.trackDataStream, (char *) &i1, sizeof (int));
for (iCluster = 0; iCluster < *nClusters; iCluster++)
  if (Clstr[iCluster].valid)
     siz = write (Pars.trackDataStream, (char *) &Clstr[iCluster],
                  sizeof (CLUSTER_INTPTS));
```

where

```
int nClusters;
  CLUSTER_INTPTS Clstr[MAXCLUSTERHITS];
```

and

```
typedef struct CL_INTPTS {
    float xx, yy, zz;
    float edet;
    int   order;   /* 0 == first interaction point */
    long long int timestamp;
    int   shellHitPos; /* internal use only */
    int   detno; /* not sure it is used or meaningful */
    } CL_INTPTS;


typedef struct CLUSTER_INTPTS {
  int   valid; /* alway 1 in output, may be zero internally */
  int   ndet; /* # interaction points */
  int   tracked; /* ==1 if we managed to track */
  float fom;  /* fom value for the tracking */
  float esum;   /* gamma ray energy */
  int   trackno;
  int   bestPermutation;
  int   processsed; /* not used now */
  CL_INTPTS intpts[MAX_NDET];
} CLUSTER_INTPTS;
```

The (xx,yy,zz) values for order==0 is the first interaction point for the γ ray as determined by the tracking code. The output data from trackMain can be read with a code like this (taken from ctkana.c)

```
  siz = read (infile, (void *) &track->n, sizeof (int));
      ptgd = track->gd;
      ptinp = track->inp;
      for (i = 0; i < track->n; i++)
        {
          siz = read (infile, (char *) ptgd, sizeof (GEBDATA));
          siz = read (infile, (char *) ptinp, ptgd->length);
          ptgd++;
          ptinp++;
        };
  siz = read (infile, (void *) &nClusters, sizeof (int));
  siz = read (infile, (void *) &Clstr[i], sizeof (CLUSTER_INTPTS));
```

C The BGS data format

This section needs to be filled out by some of the BGS people or Heather. What little I know came from I-Y Lee and is implemented in the function 'unpackBGS' found in the ctkana.c code available from http://www.phy/anl.gov/gretina

D Options for tracking

All the options for tracking are listed and documented in the chat file that the trackMain program reads in. Below is a typical listing. Lines starting with # or ; are comments

```
#echo

#-----------------------------------
# chatscript to process hits in the GRETA
# germanium shell. Self documenting.
#-----------------------------------

# maximum number of events to cluster and track
# debug parameter really...

maxevents 2000000000

##################################################
# modify the  decomposition
# energies in the tracking
##################################################

# use the CC FPGA energies or segment FPGA energies

useCCEnergy
;useSegEnergy

##################
# clustering angle
##################

#clustering angle (degrees) [aka 'alpha']
#not necessarily defined as in NIM paper!!

clusterangle 23

##################################################
# enabled detectors

enabled "1-130"

##################################################
# for retracking of data offline set this flag
##################################################

;retrack

##############################################
# FOM we assign to single hits
```

```
# so they can be on an equal playing
# field with multihits that have a finite FOM
################################################

singlesfom 0.0

################################################
# single hit range function
# single hits outside range will be assigned (large) FOM
# so they can be cut when we sort.
# this value overwrites above^^^
# given [90% interaction length +0.5cm], see pdf for documentation
################################################

#                  +-- number of datapoints following
#                  |  +-- set FOM to this value outside range
#                  |  |    +-- target to detector distance [cm]
#                  |  |    |
singlehitmaxdepth 23 1.9 18.5
0.000 0.59
0.050 0.59
0.060 0.65
0.080 0.82
0.100 1.04
0.150 1.7
0.200 2.31
0.300 3.15
0.400 3.72
0.500 4.15
0.600 4.53
0.800 5.17
1.000 5.74
1.250 6.38
1.500 6.94
2.000 7.84
3.000 9.01
4.000 9.66
5.000 10
6.000 10.16
8.000 10.17
10.00 10.01
16.3  10.01
# |   |
# |   +-- range (from surface of crystal) [cm]
#+-- energy [MeV]
# [data points for 0.0 and 16.3 must be there]

#---------------------------------------
# ndet and associated energy range limits
#---------------------------------------
```

```
# for each number of sector hits (==ndet)
# we expect a certain energy range. Assign
# events outside these limets a FOM that
# excludes them from further analysis
#        +-- ndet value
#        |    +-- elo limits
#        |    |     +-- ehi limits
#        |    |     |   +-- assign this FOM value if outside
#        |    |     |   |
ndetElim 1   0.0   0.8 1.81
ndetElim 2   0.0   1.6 1.82
ndetElim 3   0.0   3.2 1.83
ndetElim 4   0.4   6.4 1.84
ndetElim 5   0.8  18.8 1.85
ndetElim 6   1.6  27.6 1.86
ndetElim 7   3.2  53.2 1.87
ndetElim 8   6.4  99.0 1.88
ndetElim 9  18.1  99.0 1.89


#####################
# tracking strategies
#####################

#----------------
# --> specify the tracking strategies to use
# trackingstrategy <ndet> <option>
#  0: full tracking
#  1: [REMOVED] largest energy point (==first in E sort)
#  2: [REMOVED] random point
#  3: 'kickout', break @ first worse permutation
#  4: 'goodenough', break @ first OK FOM encountered
#  5: 'jump', specify jump option: g...t...

trackingstrategy 1 0
trackingstrategy 2 0
trackingstrategy 3 0

# use for full tracking (usually too costly CPU wise)
;trackingstrategy 4 0
;trackingstrategy 5 0
;trackingstrategy 6 0
;trackingstrategy 7 0
;trackingstrategy 8 0

# use for realistic realtime tracking
trackingstrategy 4 5 ggtt
trackingstrategy 5 5 ggttt
trackingstrategy 6 5 ggtttt
```

```
trackingstrategy 7 5 gggtttt
trackingstrategy 8 5 gggttttt

#-------------------------------------------------
# figure of merit cuts. In the trackingstrategy
# of 'goodenough', we bail if we get a FOM below
# the fomgoodenough value given. In the 'jump'
# trackingstrategy we jump to next group if the
# FOM is below the fomjump value given.

fomjump 1.0
fomgoodenough 1.0

#########################################################
# methods for dealings with UNTRACKED (monster) clusters
#########################################################

#----------------
# enable reclustering for UNTRACKED (monster) clusters
# [methode 1 of 2 to deal with those]
#               +- 'kickout' FOM value
#               |   +- threshold FOM for recluster
#               |   |   +- ndet minimum for reclustering
#               |   |   |  +- max number of reductions in alpha
#               |   |   |  |   +- reduction factor for alpha
#               |   |   |  |   |
recluster1   0.01  0.1  3 10  0.90

#----------------
# enable splitting of clusters for UNTRACKED (monster) clusters
# [methode 2 of 2 to deal with those]
#               +- 'kickout' FOM value
#               |   +- threshold FOM for splitting
#               |   |    +- ndet minimum for splitting
#               |   |    |  +- ndet maximum for splitting
#               |   |    |  |  +- max number of tries before giving up
#               |   |    |  |  |
#               |   |    |  |  |
splitclusters1 0.01 0.6  3 16 100

#----------------
# fom kickout value for big untrackted monster clusters
# when they are split (we can't use a fraction
# as we do for already tracked clusters)

untracked_fom_kickout 0.1


################################################
# methods for splitting already tracked clusters
```

```
# that don't look right
################################################

#----------------
# enable reclustering for TRACKED clusters
#               +- 'kickout' FOM value
#               |   +- threshold FOM for recluster
#               |   |   +- ndet minimum before trying to recluster
#               |   |   |  +- max number of reductions in alpha
#               |   |   |  |   +- reduction factor for alpha
#               |   |   |  |   |
recluster2  0.10  0.8 3  6  0.90

#----------------
# enable splitting of clusters
#                +- 'kickout' FOM value
#                |   +- threshold FOM for splitting
#                |   |   +- ndet minimum for splitting
#                |   |   |  +- ndet maximum for splitting
#                |   |   |  |  +- max number of tries before giving up
#                |   |   |  |  |   +- good enough improvement fraction
#                |   |   |  |  |   |
splitclusters2 0.10 0.7  3  16 200 0.4

################################################
# methods for combining already tracked clusters
# that don't look right
################################################

#----------------
# enable combination of clusters
#                +- 'kickout' FOM value
#                |   +- threshold FOM for combining
#                |   |   +- ndet maximum for combining
#                |   |   |  +- max distance for inclusion attempt
#                |   |   |  |
combineclusters 0.10 0.1  5 10000

###################################
# combine single hits (matchmaker)
###################################

#           +- 'kickout' FOM value
#           |    +- max distance for inclusion attempt
#           |    |
matchmaker 0.10 10000

#----------------
# number of itterative loops (more than one
# is very CPU costly! and does not help much
```

```
# at the moment, could even hurt!)

itterations 1

#----------------
# velocity v/c, eg0 is energy offset correction
# do not use!! should be removed!!!!

beta      0
eg0     0.0

#----------------
# energy gate limits (in MeV)

egat      0.05  20.0

#----------------
# number of sector hits allowed
# do not use, specified in analyze.chat

#ndetlim  2 8

#-----------------------------------
# misc

nprint 50

#-----------------------------------
# beautify the output
# (==remove invalid entries so it looks nice)

removeinvalidentries

#-----------------------------------
# suppress the non-hits in decomposed and tracked
# data.

;DOES NOT WORK YET zerosuppress

#-----------------------------------
# output file for tracked data

trackDataStream tracked.dat
```

As can be seen, there are many tracking parameters to optimize for the data set your are interested in. Because the optimal decomposition performances are not achieved yet, many of the parameters have not really been tested and optimized using in-beam real data.

E The preprocessed S800 data format

```
/*
 * S800 pre-processed (currently poor Dirk's version)
 *
 * PID plot made from tof_obje1/xfpe1/rfe1 versus ic _de
 *
 * ata/bta/dta/yta used for Doppler reconstruction
 *
 * gap = 1073mm, zfp = 0.
 * fp_afp = atan((crdc2_x - crdc1_x) / gap)
 * fp_bfp = atan((crdc2_y - crdc1_y) / gap)
 * fp_xfp = crdc1_x / 1000 + zfp * tan(fp_afp);
 * fp_yfp = crdc1_y / 1000 + zfp * tan(fp_bfp);
 *
 *
 */
#define S800PHYSDATA_NOTVALIDDATA -9999.999
#define S800PHYSDATA_TYPETAG  0xABCD1234
struct S800_physicsdata {
  int32_t type;    /* defined abcd1234 for indicating this version */
  float crdc1_x;   /* Crdc x/y positions in mm */
  float crdc1_y;
  float crdc2_x;
  float crdc2_y;
  float ic_sum;    /* ion chamber energy loss         */
  float tof_xfp;   /* TOF scintillator after A1900    */
  float tof_obj;   /* TOF scintillator in object box  */
  float rf;        /* Cyclotron RF for TOF            */
  int32_t trigger; /* Trigger register bit pattern    */
  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
  /* from here corrected values extracted from data above */
  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
  float ic_de;
  /* TOF values with TOF correction applied (from afp/crdc x) */
  float tof_xfpe1;
  float tof_obje1;
  float tof_rfe1;
  /* Trajectory information at target position calculated from
     a map and afp/bfp/xfp/yfp. New map and you need to re-calc */
  float ata; /* dispersive angle        */
  float bta; /* non-dispersive angle    */
  float dta; /* dT/T T:kinetic energy   */
  float yta; /* non-dispersive position */
};
typedef struct S800_physicsdata S800_PHYSICSDATA;
```