BRAHMS - Version 305
A Monte Carlo for a Detector at a 500/800 GeV Linear Collider

Ties Behnke*, DESY Hamburg, Grahame Blair†, Royal Holloway, Univ. of London and DESY,
Markus Elsing‡, CERN, Kristian Harder §, DESY,
Klaus Mönig¶, DESY Zeuthen, Vassilly Morgunov ‖IHEP Moskow and DESY
Martin Pohl**, ETHZ

September 23, 2002

Abstract

BRAHMS - Beph (the second letter of the hebrew alphabet) Reconstruction and Analysis Helpful Montecarlo Software - is a GEANT3 based description of the detector specified in the Technical Design Report. In this manual the structure of the program, the implementation of the detector and the user interface are described.

1 Overview

1.1 Introduction

An ECFA/DESY 1997 workshop series on the physics and detector requirements at a 500/800 GeV linear collider led to a Conceptual Design Report [1] which summarizes the findings of the study and which forms the starting point for the GEANT description coded in the program BRAHMS. The detector concept has been further developed and is more fully described in the TESLA technical design report, published in 2001. A few pages which summarize the detector description can be found at:

`http://www.hep.ph.rhbnc.ac.uk/~blair/detsim/brahms/summary.ps`

BRAHMS is written in FORTRAN and interfaces to GEANT3 [2]. The aim was to produce quickly a working detailed detector simulation to provide a basis for future developments and to cross-check the assumptions made in the fast simulations used for physics analyses. This report describes the version 1.02 and is intended to act as a catalyst for additions and modifications. It is expected that the author list will continue to grow and that this report, together with BRAHMS itself, will evolve with time. In this context, the reader is encouraged to criticise both this report and the program itself and to contribute actively to the development of the program.

As FORTRAN and GEANT3 are quickly becoming obsolete future development of a full detector simulation for a detector at a linear collider will shift more and more to GEANT4 and

*email:Ties.Behnke@desy.de
†email:blair@ppu1.ph.rhbnc.ac.uk
‡email:Markus Elsing@cern.ch
§email:Kristian.Harder@desy.de
¶email:moenig@ifh.de
‖email: Vassilly.Morgunov@desy.de
**email:pohl@ifh.de

other, more modern languages. Nevertheless for the time being BRAHMS will be maintained as the basis of the simulation software until a full replacement is available.

The user of this program is reminded that te geometry of the TDR detector is fairly complicated, and no warranty is given that the implementation in BRAHMS is correct. Care should be taken when interpreting results.

The intended user of the code will tend to be interested more in the details of detector design, or in physics studies where simple smearing is not appropriate, rather than in general physics studies. For this reason (and due to time constraints) the code has not been fully optimised for the general user.

A fast simulation program describing the same detector is available from the above mentioned WEB pages. SIMDET V4 is the most recent version and has the TDR detector fully implemented.

1.2 Getting the Code

The code is stored in a central AFS directory

```
/afs/ifh.de/group/lc
```

Three versions of the code are stored at one time – an old one, the current default one, and a development version. Sources, makefiles and examples are provided for each version. The tree structure of the depository is shown in figure 1. With access to AFS, the user can simply run

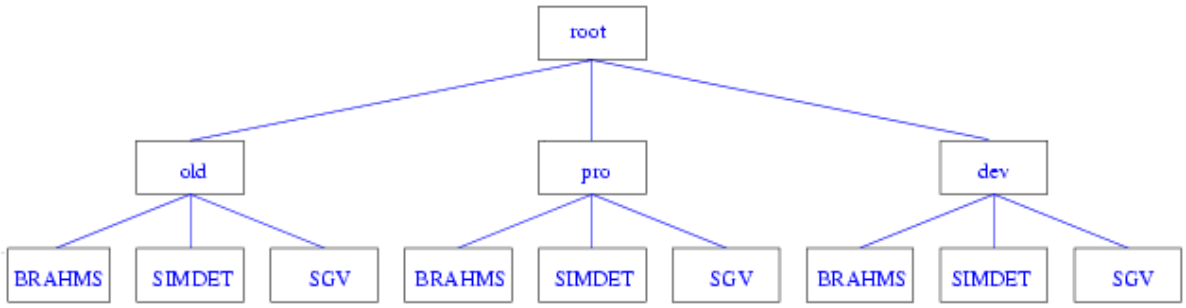

Figure 1: Layout of the central code depository /afs/ifh.de/group/lc for the `BRAHMS` program.

the makefile to obtain a running version without any need to copy the source files into local directories.

General information on the simulation programs of the ECFA / DESY workshop can be found at the web site:

```
http://www.ifh.de/linear_collider/
```

A dedicated `BRAHMS` Web page is accessible from this page.

2 The Code

2.1 General

The code is written in `FORTRAN` under the control of `PATCHY`. The code is organised in a number of different `patches`.

BRDOC	some documentation;
BRCDES	the common blocks used in BRAHMS;
BRMAINB	the main program for **B**atch running;
BRMAINI	the main program for **I**nteractive running;
BRKERN	the central routines (essentially these are the GEANT user routines);
BRGEOM	routines to define the geometry;
BRUTIL	some utilities;
BRTRAK	the track fitting code;
BRBMV1	definitions of the beam line and other machine elements as defined in the CDR;
BRBMV2	definitions of the beam line and other machine elements for the new beamline designs 1999. Either BRBMV1 **or** BRBMV2 should be used.
GECDES	Geant common blocks.

In addition, and in order to avoid people having to use the GEANT cards file, a patch GECDES is provided, in which the GEANT common blocks used in BRAHMS are duplicated.

The recosntruction code used in the tracking and in the calorimeter detectors is now kept in separte directories and files. The subdirectory `tracker` contains the tracking code, the directory `calo` the calorimeter code.

The program can be run in interactive mode or in batch mode. For debugging it is useful to store the positions along particle tracks; this also produces interesting event displays during the interactive mode and is facilitated via a call of `GSXYZ` in `GUSTEP`. You can switch between interactive and batch mode by using the appropriate patch `brmaini` or `brmainb` and doing so will also automatically set the program up in such a way that `GSXYZ` is called when the interactive mode is chosen. Note that this changes the main routine of the program – *be aware* if you make changes to this part of the code.

2.2 Structure of the program

The program is written in the context of GEANT3. The routines specific to the Linear Collider Detector are implemented via the GEANT user routines `GUxxxx`. They are explained in more detail in the following sections.

The code is split into two main parts: the simulation and the reconstruction part. If you want to can build programs for only the reconstruction or only the simulation.

To ease the interface of user analysis code with `BRAHMS` a structure of user routines is provided, which closely models the GEANT routines, and which are called at the end of the relevant GEANT routines. These routines are

UBINIT	initialisation, before FFREAD
UBSETR	initialisation, after FFREAD
UBKINE	after generator input
UBTRAK	called after GUTRAK for each GEANT track individually
UBTREV	called after GUTREV, once per event
UBSTEP	called after GUSTEP
UBOUT(i)	called at the end of each event i=1 after the end of the simulation i=2 after the end of the reconstruction

2.2.1 The simulation program

The simulation part of the program, the real BRAHMS, does all the GEANT tracking of particles and computes hits in the sensitive detector volumes. Depending on the detector not

all GEANT hits are stored, but they are converted into hits in electronic cells (e.g. in the calorimeter).

The hits are then communicated to the reconstruction program. This can be done either internally, in which case the user has not to do anything, or they can be passed via an intermediate hits file. The structure of the hits file is explained later.

2.2.2 The reconstruction program

THe reconstruction of the simulated hits is done in the second part of the program. If run in one job with the simulation the reconstruction is started from the routine GUOUT, at the end of the GEANT tracking job. If run as a standalone program the system reads in the hits from the hits file, and then starts the reconstruction. To give the reconstruction access to the detector geometry the GEANT infrastructure is used in this case as well to build a geometrical model of the detector.

NOTE: At the moment there is no mechanism to make sure that the geometry model of the reconstruction program is the one actually used at the time the hits were generated. It is the responsibility of the user to ensure that the system starts from coherent data.

2.3 User Control

The user of `BRAHMS` can control the program execution through the CERN `FFREAD` free format input package. By default both in the interactive and in the batch version a file with FFREAD cards is read in after the UBINIT, and before the UBSETR routine. All GEANT FFREAD cards are available and can be set.

The FFREAD cards file is read in through the "generic" name `cardsfile`. The user is expected to set a symbolic link from the actual file to this name.

2.3.1 BRAHMS FFREAD cards

At the time of writing the following FFREADS cards have been implemented. Contributions for more cards are welcome.

- **turnon** Accepts an array of 12 Integer values corresponding to the different subdetectors. A value of 0 switches the detector off, > 0 on. The following table shows the available switches and the order in which they have been defined. The mnemonic given is the name of the logical variable used in the BRAHMS code to control the subdetector.

1	**VTX**	=1 Pixel detector, =2 CCD detector for CCD: add 10 to get detector for 1cm beampipe; add 20 to get detector for 2 cm beampipe There is currently no APS version for a 1 cm beampipe.
2	**ITC**	Intermediate Tracker
3	**FTD**	Forward Tracking Detector
4	**SIT**	Silicon intermediate Tracker
5	**TPC**	TPC
6	**BCAL**	Barrel calorimeter
7	**ECAP**	Endcap calorimeter
8	**Coil**	The Coil
9	**Yoke**	instrumented Iron
10	**Lumi**	Luminosity calorimeter inside the mask
11	**Mask**	The Tungsten Mask
12	**Bpipe**	=1: normal beampipe; =2: extended beampipe to $\pm$100m
13	**Pre**	The preshower Detector
14	**Quad**	beamline elements (quadrupoles etc)
15	**Hall**	The walls (concrete) of the Hall and Tunnel

- **brpmin** Minimum momentum to be tracked inside the detector volume, default = 100 MeV.

- **ibrpid** Array to define which particles are to be tracked in sensitive detector volumes. Order is that of Geant.

- **kinfile** Steers the input of 4-vectors into BRAHMS.
  =1 Read 4-vectors from file "kinfile" via routine lcread
  =2 use internal routine USKINE for user supplied interface to generator, simple track generators or similar.

- **calopt** Controls the level of detail for the calorimeter simulation. Its meaning is the same as the variable `ICALOPT` which is described later in the writeup in the calorimeter section.

- **hcgap** Controls the size of an extra gap between the HCAL and the coil. Default value is 0. This variable allows for calorimeter studies to determine the effect of reducing the depth of the calorimeter inside the coil.

2.4 Input/Output

The Input is performed in the subroutine `GUKINE`. Two modes exist: the user can read 4-vectors from a file via the routine `LCREAD` in a format agreed for the workshop. Alternatively a routine `USKINE` is called, where users can implement their own event generators. At the moment an example for a simple single particle generator is included.

The kine file is read in under the "generic" name `kinfile`, which again the user is expected to set.

Output of simulated events in a common format is not yet implemented. At the moment a few simple common blocks are filled, which can be analysed by the user after the routine `GUTREV` has finished. The routine `UBTREV` is foreseen for this purpose. A more detailed definition of the common block structure is given later in this writeup.

2.5 Defining the Detector Geometry

The definition of the detector geometry happens in two steps, as usual in Geant. First the materials have to be defined, then the actual geometry. The master routine called by GEANT is UGEOM. Logically the detector is divided into four volumes:

- cave: the master volume, which encloses all other volumes.
- inner detector, volume CDET: all detector elements which are inside the coil, including the coil itself.
- outer detector, volume ODET: all detector elements which are outside the coil.
- machine, volume MACH: all detector elements which are connected to the machine: beampipe, beamline elements, etc.

The structure of this logical subdivision is mirrored by separate subroutines, which build the different detector parts. Each of these routines then in turn calls one routine for every subdetector.

2.5.1 Material Definitions

Media in Geant need to be defined in terms of the properties when interacting with different sorts of radiations. This is done in two places in the program: Materials, which are needed by more than one detector, are defined in one central place in the routine `DEFMED`. These definitions do not depend on the specifics of the detector.

Materials which are used by one detector only, which are specific to this detector, are defined in a detector routine `DETMAT` for detector `DET`. Some properties of these materials might be defined through special detector dependent parameters. If so a routine `DETGEOM` needs to be called before `DETMAT`.

At the moment all detectors inside the coil are assumed to be immersed in a constant magnetic field of $3T$. A uniform magnetic field is defined in the opposite direction for materials outside the coil used in the outer detectors, and the magnitude is given by the ratio of areas inside and outside the coil (assuming simple flux conservation, in a very naive way). Materials used in the definition of the beamline are treated in a more complicated way to properly describe the magnetic fields inside the different beamline elements. *At present there is no special treatment of the magnetic field in the end-caps.*

2.5.2 Building Detector Components

The detector geometry is defined in two steps: a routine `DETGEOM` fills the appropriate common blocks with dimensions etc. In general the relevant common blocks are `DETDIMS`. A routine `DETBLD` then builds the actual detector. As an example, if the user wants to include the TPC the following routines are needed:

- TPCGEOM: to define the dimensions etc of the TPC
- TPCMAT: defines the TPC specific materials (gas etc.)
- TPCBLD: builds the actual geometry via calls to GSVOLU, GSPOS etc.

Switching on and off of a subdetector is defined with the FFREAD cards TURNON (see above).

2.6 The Calorimeter

The calorimeter is one of the major components to benefit from detailed simulation. There are three options to determine the detail of this simulation and are flagged by the value of an integer variable `ICALOPT` in the routine `UGEOM` (FFREAD card CALOPT).

1 `ICALOPT`=0. In this case the calorimeter is turned into single towers of 'vacuum' and on entering the calorimeter the particles are stopped and their position, momentum and type are returned in the common block `/CALOBJ/`. In this way the entire inner detector can be used for detailed tracking with realistic material interactions, after which simple smearing can be applied to the four-vectors entering the calorimeter. It is intended to provide a suitable smearing routine to combine this option directly with the fast simulations; probably via direct input to the `SIMDET` package.

2 `ICALOPT`=1. In this case the calorimeter is segmented into cells (as defined in the CDR) made of absorber/scintillator material defined as a 'porridge' in `DEFMED`. The detailed geometry of the towers is built in the routine `BLDTWR` and consists of pointing trapezoids built as far as possible as 'Russian Dolls' so that daughter volumes are contained fully within mothers. Getting involved with the details of this is strictly for the experts. (But please feel free to check!). Simple smearing is defined on the output according to the resolutions defined in `UGEOM`. For the HCAL, the origins of energy deposits as arising from either electromagnetic or hadronic interactions are noted. This is so as to be of use in e/h studies etc. The conversion of the hadronic and electromagnetic components of HCAL and ECAL hits into an "energy measurement" is performed by the routine `SMRCELL`. This is a simple routine and is best understood by looking at the code; it simply smears energy deposits according to the input resolutions and adds them with appropriate weights for the e/h ratio required. This routine needs modification by calorimeter experts, so treat HCAL measurements with care at this stage!

3 `ICALOPT`=2. This is the most detailed. Individual scintillators and absorber tiles are defined as in the CDR and light output from the scintillators is added. In this way the full sampling effects are included. This option is very slow and only a few towers can be built before over-filling memory etc. The option is intended for detailed design studies and can be used as a monte-carlo 'test beam'.

The calorimeter output is unfortunately rather complicated at this stage. It is intended (*later*) to produce an output equivalent to that of the `SIMDET` package, with the same cell structure and format. However at this stage it is assumed that the user is more concerned with details of detector studies and so will wish to remain closer to the `GEANT` format. There is a simple cluster finder routine included called `CLUSTER` which produces clusters and stores them in the common block:

```
INTEGER NMXCLUS,NMXINCL
PARAMETER(NMXCLUS=100)
PARAMETER(NMXINCL=1000)
INTEGER CLUS(NMXCLUS,NMXINCL),NCINCLS(NMXCLUS),NCLUS
COMMON/CLUSCOM/CLUS,NCINCLS,NCLUS
```

where `NCLUS` is the number of clusters found, `NCINCLS(ICLUS)` is the number of cells in the cluster and `CLUS(ICLUS,ICELL)` is the cell number of the `ICELL`th cell in the cluster. The cell number locates the cell uniquely in one event. The position and properties of the cell are contained in the common block:

```
INTEGER NMXCELL
PARAMETER(NMXCELL=10000)
REAL CELL(NMXCELL,4)
INTEGER NCELL,LOCCELL(NMXCELL,5)
COMMON/CELL/CELL,LOCCELL,NCELL
```

where `CELL(ICELL,J)` contains the x, y, z position (J=1,3) and smeared energy deposit (J=4) of the cell. If you wish to use energy deposits in cells, together with the positions of cells as input to your own cluster finders then this is all the information you need.

The `LOCCELL` array contains additional topological information on the postion of the cell, namely which calorimeter ring, which ϕ-position of the tower, which cell position in the tower etc. This is sometimes hard to understand, even for those familiar with the code, so it is not discussed in detail here. The authors will be happy to explain things to those wishing to do detailed studies of cell stacking.

(As quick information for those embarking on such a study: `IRING`, the supertower segmentation, goes from 0 (ring at the centre of detector, 90^o to the beam axis) to 10 (ring at approx 30^o to the beam axis) for the BCAL with `IPHI`, the ϕ segmentation, going from 1 to 50 for positive z-position rings and from 51 to 100 for negative z-position rings. *Beware* that the x, y position of cells is 'inverted' in the negative-z region because the towers are rotated to get them to point in the opposite direction. The rings `IRING` from 11 to 16 are the endcap calorimeter with `IPHI` going from 1 to `NESUPER(IRING-10)` for positive z-position rings and `NESUPER(IRING-10+1)` to `NESUPER(2*(IRING-10))` for negative z-position rings.)

2.7 Vertex Detector Options

There are two vertex detector descriptions included; a CCD pixel device which can be built by calling

```
VXDBLD_CCD
```

or an 'active pixel' design which can be build by calling

```
VXDBLD_PIXEL
```

. The vertex detector option can be selected via the FFREAD cards as discussed above. As usual, either routine should be called within `UGEOM`. The geometry of the 'active pixel' design is described in the CDR in some detail. The geometry of the CCD design can be found in the Lund workshop summary transparencies [4]. In version BRAHMS_102 the geometry of the detector has been slightly changed to correspond to the detector presented by Chris Damerell at Hamburg in Novemeber 1998. Essentially the ladders have been extended in z to cover a larger solid angle. Some first estimates of materials for support structures etc are included. At the end of the active detector layers, additional pieces od silicon have been included to simulate the readout electronics. No material for cables etc in the foward region is included – *this should be taken into account when studying the forward region of the detector!*. This geometry should be treated as preliminary and with caution.

The VXD hits are obtained by simple spatial smearing of the energy deposits performed in the routine `GUSTEP`. They are contained in the array `VXDHITS` in the routine `GUTREV`.
IMPORTANT NOTE: Please treat this code with caution - it needs to be checked by the relevant detector experts.

The output hits from the VXD are contained in the array `VXDHITS` in the routine `GUTREV`. As with all tracking detector hits they are included in the common `/CHGTRAK/` described below.

2.8 Forward Tracking Detectors

The FTD consists of a set of five silicon detectors placed along both z-directions. The geometry is described in the CDR summary. As for the VXD, the FTD hits are obtained by simple spatial smearing of the energy deposits. They are contained in the array `FTDHITS` in the routine `GUTREV`. As with all tracking detector hits they are included in the common `/CHGTRAK/` described below.

2.9 Central Tracker - TPC

The TPC is implemented closely to that described in the CDR. At present simple, constant, $r\phi$ and z resolutions are assumed although the functionality exists for z-dependent resolutions as well as statistical effects involving the number of ionisation electrons. This is all treated in the routine `GUSTEP` and the hits stored in the array `TPCHITS`. As with all tracking detector hits they are included in the common `/CHGTRAK/` described below.

The TPC is segmented into 118 radial rings and then smeared in $r\phi$. The z-position is also smeared according to the input values in `UGEOM`.

2.10 Inner Tracker - ITC

The ITC has been included as a mini-TPC by Klaus Mönig. Dimensions are as given for the straw tracker in the CDR. A readout structure similar to that described for the TPC is provided.

2.11 Information stored for tracks

In the following the documentation given for the different structures which are used to store hits and tracks during the tracking are repeated. They are a straight lift from the code- the reader is advised to check in the code that the version repeated here is up-to-date.

```
************************************************************************
* TRACKING SUBSYSTEM: PATTERN RECOGNITION AND TRACK MERGING
*
* TKBANK data structure
*
* Kristian Harder, 30 Sep, 1999
*
* many things taken from DELPHI TANAGRA manual by D. Bertrand, L.Pape
************************************************************************

************************************************************************
* a) MC track bank: true track parameters for all tracks in the detector
************************************************************************
*
*    This bank is filled by the TK package before calling the local patrec
*    modules.
*
* --- total number of tracks that produced at least one hit

INTEGER TKNTRK

* --- track information (equivalent arrays for integer and real access)
*     first index:
```

```
*       1 (real)    px
*       2 (real)    py
*       3 (real)    pz
*       4 (real)    E
*       5 (real)    x of track origin
*       6 (real)    y of track origin
*       7 (real)    z of track origin
*       8 (integer) GEANT particle code
*       9 (integer) GEANT track number
*      10 (integer) number of hits left in detector
*      11 (integer) HEPEVT track number (if any, zero otherwise)
```

The information is accessible through two statement functions, called

- TKMCTR(11,NTKMX) (real function)
- ITKMCT(11,NTKMX) (integer function)

```
************************************************************************
* b) HIT bank: all hits collected in current event
************************************************************************
*
*    All hits recorded in the current event are saved here for use
*    of the local pattern recognition modules. Filled by the TK package
*    before the local patrec modules are called.
*
*
* --- total number of hits

      INTEGER NTKHIT

* --- hit information (equivalent arrays for integer and real access)
*     first index:
*       1 (real)    x
*       2 (real)    y
*       3 (real)    z
*       4 (real)    energy
*       5 (integer) subdetector ID
*       6 (integer) track ID, 0 for real data (sic!)
*       7 (integer) pointer to first exclusion (0 if none)
*       8 (integer) number of exclusions
*       9 (integer) resolution code (see below)
*      10 (real)    resolution 1
*      11 (real)    resolution 2
*     position 7 and 8 are used for mutually exclusive hit candidates
*     (mirror hits etc.). they point to entries in the exclusion list
*     defined below.
*
*     resolution code: two bits are set. The lower value bit describes which
```

```
*                     resolution is described by resolution 1 (pos. 10), the
*                     higher value bit describes resolution 2 (pos.11).
*
*                     bit  0  - error on r*phi
*                     bit  1  - error on z
*                     bit  2  - error on r
```

The hits are available as two statement functions:

- RTKHIT(MXHT,NHMAX) (real function)
- ITKHIT(MXHT,NHMAX) (integer function)

```
************************************************************************
*
* --- exclusion list (shared by all other banks)

      INTEGER NEXCL,IEXCL(MXEXCL),EXCLERR

* --- pointer array: IHPOINT(ID_DET) points to position of first hit
*     from subdetector ID_DET. IHNUMB(ID_DET) stores the number of hits
*     found in this detector

      INTEGER IHPOINT(3000),IHNUMB(3000)


************************************************************************
* c) TE bank (subdetector output)
************************************************************************
*
*
*    Following DELPHI nomenclature, the information delivered by a subdetector
*    is called track element (TE). A TE may contain fitted tracks (e.g. TPC),
*    but may also contain single hits (e.g. Presampler hits).
*    This bank must be filled by the local patrec modules.
*
* --- number of track elements (TEs).

      INTEGER NTE

* --- DELPHI TANAGRA TE structure for track elements/hits (equivalant arrays)
*             first index (m=length of packed covariance matrix):
*              1 (integer) subdetector ID
*              2 (integer) submodule: bit code for VTX/FTD layers used in TE
*              3 (integer) not used (yet) - MUST be set to zero
*              4 (integer) measurement code (see below)
*              5 (integer) pointer to end of TE (m+17)
*              6 (integer) charge (0=neutral, 1=positive, 2=negative, 3=unknown)
*              7 (integer) number of degrees of freedom
```

```
*             8 (real)    chi^2 of the fit
*             9 (real)    length of the track element
*            10 (real)    coordinate 1 of reference point: x or R
*            11 (real)    coordinate 2 of reference point: y or R*PHI
*            12 (real)    coordinate 3 of reference point: z
*            13 (real)    theta angle
*            14 (real)    phi angle
*            15 (real)    1/p at the reference point (in 1/(GeV/c))
*                         (or 1/p_t - see measurement code)
*            16 (real)    dE/dx (if measured - see measurement code)
*            17 (real)    covariance matrix for measured quantities
*            ...          (see below)
*           m+16 (real)  error on dE/dx if dE/dx is measured
*           m+17 (real)  must be 0.0
*
*           measurement code:
*           bit  0 = 0 for x,y,z coordinates
*                  = 1 for R,R*PHI,z coordinates
*           bit  1 = 0
*           bit  2 = 1 if coordinate 1 is known (index 10), 0 otherwise
*           bit  3 = 1 if coordinate 2 is known (index 11), 0 otherwise
*           bit  4 = 1 if coordinate 3 is known (index 12), 0 otherwise
*           ('known' means either defined by detector geometry or measured)
*           bit  5 = 1 if R*PHI (x) is measured for bit 0 = 1 (0)
*           bit  6 = 1 if z (y) is measured for bit 0 = 1 (0)
*           bit  7 = 1 if theta is measured
*           bit  8 = 1 if phi is measured
*           bit  9 = 1 if 1/p is measured
*           bit 10 = 1 if 1/p_t is measured
*           ('measured' really means measured, i.e. this quantity contributes
*            to the covariance matrix)
*           bit 11 = 0
*           bit 12 = 0
*           bit 13 = 1 if dE/dx is measured
*           bit 14-... = 0
*
*           covariance matrix:
*           every variable activated by a 1 in bit 5..10 is included
*           in the covariance matrix. for a 3d tracking detector like
*           a TPC, the measured quantities are in most cases
*
*           R*PHI and z (coordinates of the reference point at a reference
*                        radius R which therefore has no error itself)
*           theta,phi,1/p
*
*           In this example, the order of elements in the covariance matrix
*           is the following:
*
*           D(R*PHI,R*PHI),
*           C(R*PHI,z),     D(z,z),
```

```
*             C(R*PHI,theta), C(z,theta), D(theta,theta),
*             C(R*PHI,phi),   C(z,phi),   C(theta,phi),   D(phi,phi),
*             C(R*PHI,1/p),   C(z,1/p),   C(theta,1/p),   C(phi,1/p), D(1/p,1/p)
```

The TE are accessble through statement functions:

- ITE(MXTE,MXEVTE) (integer function)
- REAL RTE(MXTE,MXEVTE) (real function)

```
* --- additional information. first index:
*      1 (integer) position of first associated hit in hit list (see below)
*      2 (integer) number of hits
*      3 (integer) pointer to first exclusion (0 if none) in exclusion list
*      4 (integer) number of exclusions
*      5 (integer) track no. (positive if 95% of the hits belong to
*                             same track, negative if 75%, zero below.)
*     remark: the exclusion list is shared with the hit bank (see there).
```

This information is accessible through a statement function:

- ITEDAT(5,MXEVTE) (integer function)

```
* --- hit list: pointers to entries in the hit bank.

      INTEGER NHITTE,IHITTE(NHMAX)
```

```
**********************************************************************
* d) TS bank (ambiguous full track candidates)
**********************************************************************
*
*    Merging result as created by the TK package and DELSEARCH.
*    Ambiguous use of TEs is possible here; the unambiguous optimal
*    track reconstruction will be entered into the TK bank (see below)
*
* --- number of tracks segments (TS) found by DELSEARCH

      INTEGER NTS

* --- DELPHI TANAGRA TS structure
*             first index (m=length of packed covariance matrix):
*              1 (integer) module identifier (internal use only)
*              2 (integer) bit code for used detectors
*              3 (integer) measurement code (see below)
*              4 (integer) track type (see below)
*              5 (integer) number of TEs used in this TS
*              6 (integer) charge (0=neutral, 1=positive, 2=negative, 3=unknown)
```

```
*             7 (integer) not used (must be zero, though)
*             8 (integer) number of degrees of freedom of the track fit
*             9 (real)    chi^2 of the track fit
*            10 (real)    track segment length
*            11 (real)    x of TS start point
*            12 (real)    y of TS start point
*            13 (real)    z of TS start point
*            14 (real)    x of TS end point
*            15 (real)    y of TS end point
*            16 (real)    z of TS end point
*            17 (real)    coordinate 1 of reference point: x or R
*            18 (real)    coordinate 2 of reference point: y or R*PHI
*            19 (real)    coordinate 3 of reference point: z
*            20 (real)    theta angle
*            21 (real)    phi angle
*            22 (real)    1/p at the reference point (in 1/(GeV/c))
*                         (signed with geometric curvature of track
*                          measured from nside to outside of detector)
*            23 (real)    covariance matrix for measured quantities
*            ...          (as above)
*            37 (real)
*
*            measurement code:
*            bit  0 = 0 no estimate at all for the track segment parameters
*                   = 1 track segment parameters are given
*            bit  1 = 0 for x,y,z coordinates
*                   = 1 for R,R*PHI,z coordinates
*            bit  2 = 0 crude estimation of errors (diagonal cov. matrix)
*                   = 1 full covariance matrix
*            bit  3 = 0 all parameters are given if bit 0 is on
*                   = 1 no error matrix/starting point/end point
*
*            track type:
*            not yet documented. 1+8 (bits 0 and 3 set) means:
*            crude estimation, plane reference surface (cartesian coordinates),
*            no start/end points given

      INTEGER ITS(MXTS,NTSMX)
      REAL    RTS(MXTS,NTSMX)
      EQUIVALENCE (ITS,RTS)

* --- additional TS information. first index:
*      1 (integer) position of first associated TE in TE list (see below)
*      2 (integer) number of TEs
*      3 (integer) pointer to first exclusion (0 if none) in exclusion list
*      4 (integer) number of exclusions
*      5 (integer) track no. (positive if 95% of the hits belong to
*                             same track, negative if 75%, zero below.)
*     remark: the exclusion list is shared with the hit bank (see there).
```

```
      INTEGER ITSDAT(5,NTSMX)

* --- TE list. contains all TEs that are used by the TS. (referenced in ITSDAT)

      INTEGER NTSTEL,ITSTEL(MXEVTE)

************************************************************************
* e) TK bank (the real tracking result - unambiguous track candidates)
************************************************************************
*
*    Filled by TK package after resolving all TS ambiguities.
*
*
* --- number of tracks (TK)

      INTEGER NTK

* --- DELPHI TANAGRA TK structure
*             first index (m=length of packed covariance matrix):
*              1 (integer) module identifier (internal use only)
*              2 (integer) bit code for used detectors
*              3 (integer) measurement code (see below)
*              4 (integer) track type (as above)
*              5 (integer) number of TEs used in this TK
*              6 (integer) charge (0=neutral, 1=positive, 2=negative, 3=unknown)
*              7 (integer) not used (must be zero, though)
*              8 (integer) number of degrees of freedom of the track fit
*              9 (real)    chi^2 of the track fit
*             10 (real)    track length
*             11 (real)    x of TS start point
*             12 (real)    y of TS start point
*             13 (real)    z of TS start point
*             14 (real)    x of TS end point
*             15 (real)    y of TS end point
*             16 (real)    z of TS end point
*             17 (real)    coordinate 1 of reference point: x or R
*             18 (real)    coordinate 2 of reference point: y or R*PHI
*             19 (real)    coordinate 3 of reference point: z
*             20 (real)    theta angle
*             21 (real)    phi angle
*             22 (real)    1/p at the reference point (in 1/(GeV/c))
*                          (signed with geometric curvature of track
*                           measured from inside to outside of detector)
*             23 (real)    covariance matrix for measured quantities
*             ...          (as above)
*             37 (real)
*
*             measurement code:
*             bit  0 = 0 for x,y,z coordinates
*                    = 1 for R,R*PHI,z coordinates
```

```
      INTEGER ITK(MXTK,NTKMX)
      REAL    RTK(MXTK,NTKMX)
      EQUIVALENCE (ITK,RTK)

* --- additional TK information. first index:
*      1 (integer) position of first associated TE in TE list (see below)
*      2 (integer) number of TEs
*      3 (integer) track no. (positive if 95% of the hits belong to
*                             same track, negative if 75%, zero below.)

      INTEGER ITKDAT(3,NTKMX)

* --- TE list. contains all TEs that are used by the TK. (referenced in ITKDAT)

      INTEGER NTKTEL,ITKTEL(MXEVTE)



**********************************************************************

C --- the tracking common block
      COMMON /TKBANK/ TKNTRK,
*     &  TKMCTR,
     >                NTKHIT,
*     &  ITKHIT,
     >                NEXCL,IEXCL,EXCLERR,
     >                IHPOINT,IHNUMB,
     >                NTE,
*     ITE,ITEDAT,
     >                NHITTE,IHITTE,
     >                NTS,ITS,ITSDAT,NTSTEL,ITSTEL,
     >                NTK,ITK,ITKDAT,NTKTEL,ITKTEL
      SAVE   /TKBANK/

**********************************************************************

+KEEP,TKNOISEPAR.
*********************************************************
*
* background hits per bunch crossing for tracking detectors
*
*********************************************************

      INTEGER TNVXL1,TNVXL2,TNVXL3,TNVXL4,TNVXL5
      INTEGER TNVTX1,TNVTX2,TNVTX3,TNVF2A
      INTEGER TNFTD1,TNFTD2,TNFTD3,TNFTD4,TNFTD5,TNFTD6,TNFTD7
      INTEGER TNSIT1,TNSIT2
      INTEGER TNFCH
      INTEGER TNNBX,TNENERGY
```

```
 COMMON /TKNOISEPAR/
>        TNVXL1,TNVXL2,TNVXL3,TNVXL4,TNVXL5,
>        TNVTX1,TNVTX2,TNVTX3,TNVF2A,
>        TNFTD1,TNFTD2,TNFTD3,TNFTD4,TNFTD5,TNFTD6,TNFTD7,
>        TNSIT1,TNSIT2,
>        TNFCH,
>        TNNBX,TNENERGY
 SAVE /TKNOISEPAR/
```

2.12 Track Reconstruction

A track reconstruction routine has been implemented based on a Kalman Filter helix fit package [5] developed for the DELPHI experiment. No pattern recognition is performed, instead the Monte Carlo track is taken as a starting point.

The track reconstruction is done in two steps. In a first step the hits in the TPC and separately the hits in the ITC are fitted to reconstruct the track segments measured in both detectors. In the second step these track segments are combined with the silicon hits to reconstruct the full track. The first step can later be replaced by a local pattern recognition for the two subdetectors, the second step can be interfaced after a future global track search.

The Kalman filter track fit uses a simplified description of the detector material to allow for energy loss and multiple scattering. The material is represented in list of surfaces, which are either symmetric cylinders or planes around the beam pipe. The energy loss in GeV (for a mip) and the number of radiation length, for a particle crossing the surface perpendicular, are stored in the structure. Two setups are prepared in the current version representing the CCD or the PIXEL setup of the detector.

A second set of surfaces is implemented containing at least one surface per subdetector. The fit package generates extrapolations to all of these surfaces which are crossed by the fitted track. These extrapolations contain the full energy loss and multiple scattering corrections. Such improved extrapolations (compared to a simple helix ansatz) are very useful for a future pattern recognition package. They are not yet used in the code.

The fit package contains an outlayer removal on the basis of a χ^2 cut. This logic can be used inside a future pattern recognition algorithm to remove fake hits associated to a track.

The name of the fit package is `DELFIT`. It is called via the routine:

```
 SUBROUTINE TRKFIT(NHITS,TRKHITS,VPOS,FITRES,FITCOV,
&                  CHI2,IERROR,NDF)
```

The parameters are:

input:

NHITS		Number of hits on the track
TRKHITS		(NMXCHIT,5) dimensional array containing hit values:
	(1)	x-position of hit
	(2)	y-position of hit
	(3)	z-position of hit
	(4)	number of ionized electrons in hit
	(5)	Type of detector hit
	(6)	$r\phi$ error of hit
	(7)	z error of hit
VPOS		3-Dimensional vector for reference point

Output:

FITRES		Fit result $(\sigma_{r-\Phi}, \sigma_z, \theta, \phi, 1/p)$
		$(\sigma_{r-\Phi}, \sigma_z$ are the impact paramaters wrt. the reference point.)
FITCOV		packed covariance matrix
chi2		fit chi2
TRKHITS	(5)	on output TRKHITS(5) is set negative if the hit was rejected by the outlier removal
ndf		number degrees of freedom of the fit after removing outlyers.
IERROR		Error flag; ZERO if OK.

`TRKFIT` converts the hits into the internal DELPHI format and steers the fitting of the track-elements within the TPC and the ITC and the fitting of the full track. The interface to the DELPHI fit package itself is provided by the routine `TK2FIT`
`TK2FIT` first obtains a rough estimate of the track parameters as a starting point for the fit. This is done either by using the track segment parameters from the previous fit to the TPC or ITC hits, or by preforming a polar inversion on all hits. The track fit steering routine FK3TRK is called twice to use the result of the first call to improve this starting point. After the iteration the optimal track parameters are returned as well as a list of outlayer hits removed from the fit.

2.13 Luminosity Calorimeter

The Luminosity Calorimeter is implemented as a cone of ECAL material (lead, scintillator porridge) with a geometry as outlined on p.248 of the CDR, which is reproduced in the summary web pages at the above address. The output is smeared in energy and in r, and $r\phi$ and contained in the common `LCHITS`.

```
INTEGER NLCMAX
PARAMETER(NLCMAX=100)
INTEGER ILCTRAK(NLCMAX),NUMBVLC(NLCMAX),NLCHITS
REAL LCR(NLCMAX),LCPHI(NLCMAX),LCENG(NLCMAX),LCHITS(3,NLCMAX)
COMMON/LCHITS/NLCHITS,LCR,LCPHI,LCENG,ILCTRAK
```

where `NLCHITS` is the number of hits, `LCR` is the radius of the hit, `LCPHI` is the ϕ of the hit, `LCENG` is the smeared energy and `ILCTRAK` is the GEANT track which produced the hit.

2.14 Instrumented Coil

The coil is treated as a cylinder of 'sensitive' aluminium material in the routine `COILMAT` so that the amount of energy lost in the coil can be obtained. In addition a cylindrical layer of

scintillator is included at the outside of the coil to act as a coil 'calorimeter'. The output is included in the common COILHITS:

```
      INTEGER NCLMAX
      PARAMETER(NCLMAX=1000)
      REAL ECLMAT,ECLSCN,ECLMTOT,ECLSTOT
      INTEGER ICLTRAK,NUMBVCL,NCLMAT,NCLSCN
      COMMON /COILHITS/ECLMAT(4,NCLMAX),ECLSCN(4,NCLMAX),ICLTRAK(NCLMAX),
     &                 NUMBVCL(NCLMAX),NCLMAT,NCLSCN,ECLMTOT,ECLSTOT
```

where NCLHITS is the number of coil hits, ECLMAT(4,IHIT),ECLSCN(4,IHIT) are the energy deposits and ECLMAT(I,IHIT),ECLSCN(I,IHIT) I=1,3 are the x,y,z coordinates of the deposits in the MATerial and SCiNtillator respectively. ICLTRAK is the GEANT track which produced the hit. The *total* energies deposited in the coil material and scintillator are returned in the variables ECLMTOT,ECLSTOT respectively.

2.15 Instrumented Iron

The return yoke can be built by calling YOKEBLD in UGEOM. The yoke exists as material only with any total energy deposit being output within the array YOKHITS in GUTREV. It can thus be used as a 'tailcatcher' but has no segmentation and hence is only really useful, in its present form, as a veto.

3 The HITS storage

BRAHMS offers the user the possibility to store the GEANT hits in a file and to read them at a later time into the reconstruction program. This saves significant amounts of CPU time if e.g. different reconstruction algorithms should be tested or compared.

The hits are written into a plain c-like output file, which is written using the cio-package (an implementation using the GZIO package is under development). No ZEBRA structures are used in the file. This makes it easy to read the hits structures with other programs which are not based on ZEBRA or are not written in FORTRAN.

The structure of the file is as follows:

```
------------------------------------------------------------------------
SOF      =>  Start of file marker                                16 bytes
       (1)    = 'S'
       (2)    = 'O'
       (3)    = 'F'
       (4)    = number of words in the extended header
------------------------------------------------------------------------
extended header                                  #(4)*4 bytes- 16 bytes
       (1) - (20)   code (see BRAHMS) for subdetector configuration of
                    the particular run
       (21)         number of events skipped when doing the simulation
       (23) - #(4)  not used
------------------------------------------------------------------------

SOR     =>  Start of Run marker                                  12 bytes
------------------------------------------------------------------------
|  Run header                                                    40 bytes
|
|       (1)    = run number
|       (2)    = event number
|       (3-10) = not used
|-----------------------------------------------------------------------

for each detector stored and for each event:
        ---------------------------------------------------------------
        |  Detector header                                       12 bytes
        |     (1)   = detector ID
        |     (2)   = number of detector blocks
        |     (3-10)= not used
        |--------------------------------------------------------------

for each detector block
                ------------------------------------------------------
                |  detector block header                    40 bytes
                |    (1)   = detector ID (maybe sub-ID, if implemented)
                |    (2)   = version number
                |    (3)   = # of hits stored
                |    (4)   = # of words / hit stored
                |-----------------------------------------------------

for each hit
                       ------------------------------------------------
                       | hit information
                       |           (# of hits) * (# of words) * 4 bytes
                       |-----------------------------------------------
                       |the following two pieces of information are
                       |stored for all detector EXCEPT the calorimeter:
```

```
                        |_____________________________________________
                        | MC track information
                        |                          (# of hits) * 4 bytes
                        |_____________________________________________
                        | volume information (volume number of the hit)
                        |           (# of hits) * (# of words) * 4 bytes
                        |_____________________________________________

at the end of each event

        ---------------------------------------------------------------
        | EOB        = End of Block                            12 bytes
        |_______________________________________________________________


at the end of the file
-------------------------------------------------------------------------
EOF     = End of File Marker
-------------------------------------------------------------------------
```

The information stored for each detector is different from detector and detector and mirrors at the moment the information stored in the GEANT hits common blocks. Typically the first three words are the spatial coordinates of the hit (x,y,z) in carthesian coordinates, followed by (if appropriate) the energy deposited and possibly other information. Please refer to the code to find out the exact details.

4 Building the code

To build (patchy, compile and link) the `BRAHMS` code the following things are needed:

- The source code `BRAHMS_102.car`
- The cradle `brahms.cra`
- The sourse and cradle for the track fit `DELFIT_102.car` and `delfit.cra`
- An installation with the CERN libraries

A sample brahms-cradle file might look like this:

```
+opt,m.
+exe.
+use,type.
+use,brcdes,t=e.
+use,brkern,brgeom,brutil,brtrak.
+use,brbmv1.
* to use the new beamline layout comment out the line above
* and activate the patch brbmv2.
* +use,brbmv2
* to use beahms in the interactive mode, replace
* brmainb with brmaini
```

```
+use,brmainb.
+use,bruser.
+use,FKCDES
+use,gecdes,t=e.
+pam,11,r=tkfbcde,t=attach,t=cards. delfit
+pam,11,t=attach,t=cards. brcar
+pam,11,t=attach,t=cards. bruser.car
+quit.
```

and for delfit:

```
+opt,m.
+exe.
+use,delfit
+pam,11,t=attach,t=cards. delfit.car
+quit.
```

In this cradle `brcar` stands for the brahms cards file, which is accessed via a symbolic link. If you use the sample makefile provided, this link is generated automatically. `bruser.car` is the user collection of routines, and includes the user hook routines `UBxxxx`.

The switch `TYPE` at the beginning of the cradle enables strong typing in the GEANT common blocks, and is mandatory for the BRAHMS code.

A sample makefile is available as well, which has the following form:

```
FFLAGS := -g

SOURCE = ${HOME}/brahms/
DIRECT = ${HOME}/scratch/
BRAHMS_VERSION = 102
DELFIT_VERSION = 100
CERNLB = -L/cern/pro/lib
USRLB = -L/usr/lib/
X11LB = -L/usr/X11R6/lib/
LIB   = -L/lib
COMP  = g77 -g -w

${DIRECT}brahms.run:${DIRECT}brahms.o ${DIRECT}bruser.o ${DIRECT}delfit.o
          rm -f ${DIRECT}brahms.run
          rm -f brahms.run
          ${COMP} \
                          ${DIRECT}brahms.o\
                          ${DIRECT}delfit.o\
                          ${DIRECT}bruser.o\
                          ${CERNLB}\
                          `cernlib geant jetset74 pawlib graflib genlib` \
                          ${X11LB}\
                           -lX11 \
                          ${CERNLB}\
                          -lphtools\
```

```
                        -lkernlib -lpacklib -lmathlib\
                        ${USRLB} -lm -lcrypt\
                        -o brahms.run

${DIRECT}brahms.o:${DIRECT}brahms.f
          rm -f ${DIRECT}brahms.run
          rm -f brahms.run
          ${COMP} -c ${DIRECT}brahms.f \
                        -o ${DIRECT}brahms.o

${DIRECT}delfit.o:${DIRECT}delfit.f
          rm -f ${DIRECT}brahms.run
          rm -f brahms.run
          ${COMP} -c ${DIRECT}delfit.f \
                        -o ${DIRECT}delfit.o

${DIRECT}bruser.o:${DIRECT}bruser.f
          rm -f ${DIRECT}brahms.run
          rm -f brahms.run
          ${COMP} -c ${DIRECT}bruser.f \
                        -o ${DIRECT}bruser.o

${DIRECT}brahms.f:brahms_${BRAHMS_VERSION}.car delfit_${DELFIT_VERSION}.car brahms_${BRAHMS
          rm -f y.lis
          rm -f ${DIRECT}brahms.run
          rm -f ${DIRECT}brahms.f
          ln -s -f brahms_${BRAHMS_VERSION}.car brcar.car
          ln -s -f delfit_${DELFIT_VERSION}.car delfit.car
          ln -s -f ${DIRECT}brahms.f brahms.f
          ypatchy - brahms.f brahms_${BRAHMS_VERSION}.cra .go
#         rm -f brahms.f
          rm -f brcar.car
          tail y.lis

${DIRECT}delfit.f:delfit_${DELFIT_VERSION}.car delfit.cra
          rm -f y.lis
          rm -f ${DIRECT}brahms.run
          rm -f ${DIRECT}delfit.f
          rm -f delfit.car
          ln -s -f delfit_${DELFIT_VERSION}.car delfit.car
          ln -s -f ${DIRECT}delfit.f delfit.f
          ypatchy - delfit.f delfit.cra .go
#         rm -f delfit.f
          tail y.lis

${DIRECT}bruser.f:brahms_${BRAHMS_VERSION}.car bruser.cra bruser.car
          rm -f y.lis
          rm -f ${DIRECT}brahms.run
          rm -f ${DIRECT}bruser.f
          ln -s -f brahms_${BRAHMS_VERSION}.car brcar.car
```

```
          ln -s -f ${DIRECT}bruser.f bruser.f
          ypatchy - bruser.f bruser.cra .go
#         rm -f bruser.f
          rm -f brcar.car
          tail y.lis
```

The library `lcrypt` is needed on linux systems, and can be removed on other UNIX machines. The user has to replace the appropriate directory and version number at the beginning of the makefile. Depending on the system the fortran compiler g77 might have to be replaced by the appropriate command. Typing "make" should then automatically build the program.

4.1 References

References

[1] Conceptual Design of a 500 GeV e^+e^- Linear Collider with Integrated X-ray Laser Facility. Vol 1., DESY 1997-048. Editors: R Brinkman, G Materlik, J Rossbach, A Wagner.

[2] `GEANT` Detector Description and Simulation Tool, CERN Program Library Long Writeup W5013, S.Giani *et al.*.

[3] Fiber technology applications for a future e^+e^- linear collider detector. H Leich,R Nahnhauer,R Shanidze. Physics/9801004.

[4] Flavour ID and possible improvements of LC, P.Burrows,

`http://www.quark.lu.se/workshop/ECFA/slides/default.htm`

[5] Reference to the DELPHI Kalman filter fit program