Brahms - Version 308
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[**], University of Geneva and CERN, Harald Vogt[††], DESY

February 4, 2004

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 TESLA Technical Design Report. In this manual the structure of the program, the implementation of the detector and the user interface are described.

[*] 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:Martin.Pohl@cern.ch
[††] email:harald.vogt@desy.de

Contents

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 [?] 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. Brahms is an implementation of the TDR detector. The detectors in V308 reflect the state of developments as of summer 2002.

`Brahms` is written in `FORTRAN` and interfaces to `GEANT3` [?]. 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 108 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 other, more modern languages. the next generation GEANT4 based simulation program within the european linear collider study is MOKKA, developed under the leadership of a group from the Ecole Polytechnique, France.

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 the 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 since version 308 maintained in a CVS depository in DESY Zeuthen. Access to the code is via a WEB interface to the depository, or directly through a cvs server. The code is accessible from the simulation group WEB page:

`http://www-zeuthen.desy.de/linear_collider/`

where also more instruction on buildung the code etc. may be found.

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 ??.

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.

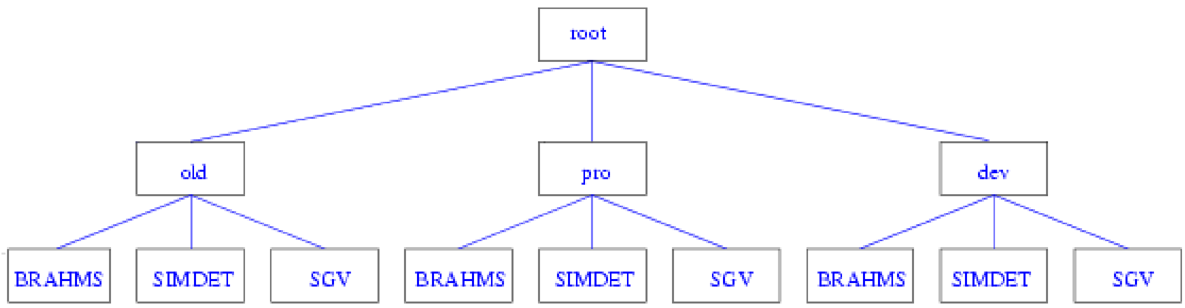

Figure 1: Layout of the central code depository for the `Brahms` program.

2 The Code

2.1 General

The code is written in `FORTRAN`. Since version 308 the code manager PATCHY has been discontinued, and instead the normal C preprocessor commands are used. The code is organised in a CVS based module Brahms. Its directory structure is shown in figure **??**.

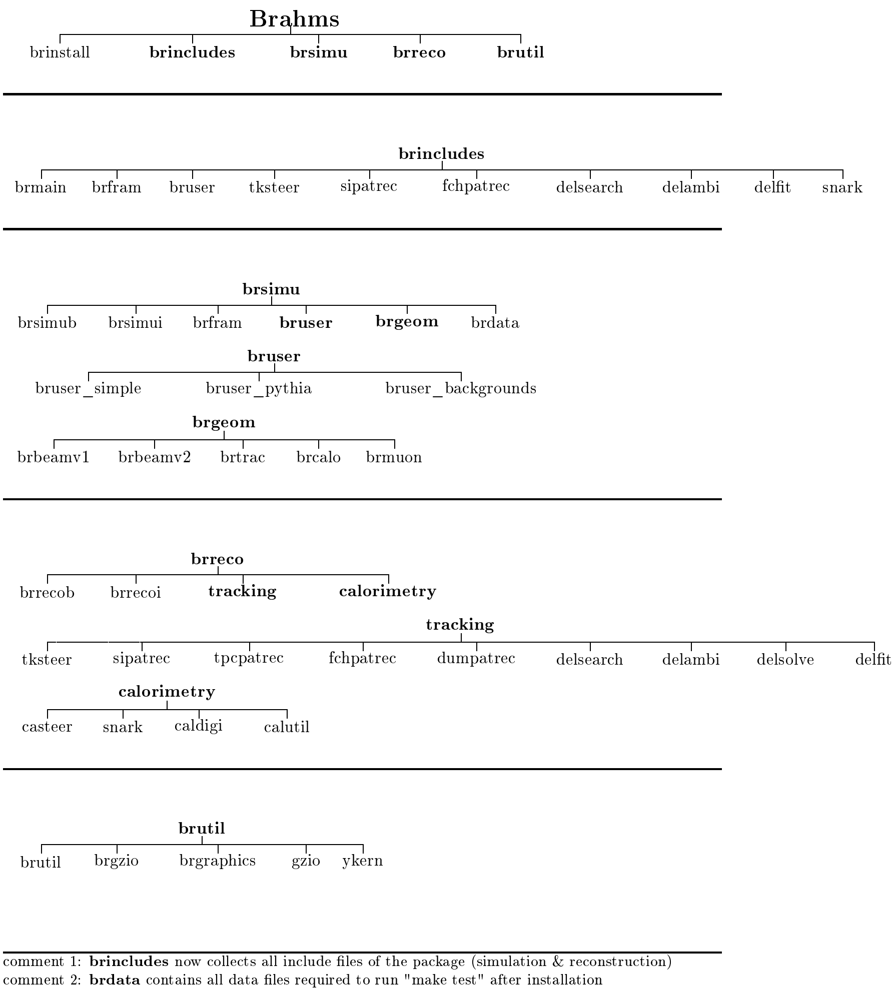

Figure 2: Layout of directory structure of the CVS module `Brahms`.

brinstall	the installation scripts for Brahms;
brincludes	the common blocks and global definitions used in Brahms;
brsimu	routines for the simulation part;
brreco	routines for the reconstruction;
brutil	some utilities;
brmainb	the main program for **b**atch running;
brmaini	the main program for **i**nteractive running;
brrecob	the main program for the reconstuction only **b**atch running;
brrecoi	the main program for the reconstuction only **i**nteractive running;
brdata	example data files for Brahms;
brgeom	routines to define the geometry; brbeamv1: definitions of the beam line and other machine elements as defined in the CDR; brbeamv2: definitions of the beam line and other machine elements for the new beamline designs 1999;
tracking	the track fitting code;
calorimetry	calorimeter code;
bruser	example code for the users interface (event generation);

All required GEANT include files used in Brahms are incorporated in the brincludes/brfram directory.

The reconstruction code used in the tracking and in the calorimeter detectors is now kept in separte directories and files. The subdirectory `tracking` contains the tracking code, the directory `calorimetry` 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 select the interactive and batch mode by using running the executables `brmaini.bin` or `brmainb.bin` and doing so will also automatically set the program up in such a way that `GSXYZ` is called when the interactive mode is chosen.

The simulation and the reconstruction code is maintained in different directories. The reconstruction code can be run independently from the simulation code. The simulation code always contains the reconstruction code, though through use of a control card it can be switched on or off at execution time. The corresponding executables `brrecoi.bin` or `brrecob.bin` will be also created by the installation.

2.2 Installation of Brahms

The installation is based on a configuration script for Brahms and two makefile templates one for the creation of the libraries and one for the creation of the executables. These files are stored in the subdirectory `brinstall`. The installation has to be done in the "root" directory `Brahms`.

One has to start with the configuration by "`./brinstall/configure`". Using the `-h` or `--help` option configure shows all its options which may be used to configure your Brahms installation. After the configuration one has to run "`gmake`" and optional "`gmake test`" to check the installation. The libraries and axecutables are stored in a seperate subdirectory (default: a subdirectory "`build`" created in the "root" directory - see the `--prefix` option of configure).

The results of `gmake test` are also stored herein. Details for the installation and possible modifications are described at the WEB page:

`http://www-zeuthen.desy.de/lc_repository/detector_simulation/dev/Brahms`

2.3 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

2.3.1 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.

2.3.2 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.

2.4 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.

2.4.1 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.

2.5 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, an example for using PYTHIA, and an example for running background files is included.

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

Simulated events are passed to the reconstruction program or written to file. The format of the intermediate hit storage is explained later in this document.

Reconstructed events are written to file using a format developed and described in the context of the SIMDET fast simulation program. The exact file format and the definition of the different quantities saved are given later in this document.

2.6 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.

2.6.1 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.*

2.6.2 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).

2.7 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).

In the Brahms release 300 and higher significant changes have been introduced for the handling and storage of calorimeter hits. The calorimeter geometry is now the one described in the TESLA TDR. The baseline option is using a very finely segmented SI-W calorimeter as electromagnetic calorimeter, and a scintillator - Iron tile calorimeter as the HCAL. Currently the alternative option of using a very finely segmented hadronic calorimeter with purely digital readout (cell hit or not hit) is not (yet) implemented in Brahms.

The calorimeters are simulated with cells of approximatley $1 \times 1cm^2$ cells, even in the HCAL. The cells are then recombined into more realistic cells in the reconstruction step. Currently only one version exists, which simulates the analogue HCAL with cell sizes of $5 \times 5cm^2$ and larger.

For a detailed description of the calorimeter simulation and reconstruction the reader is referred to [?], which is included in this note as an appendix. Means of accessing the calorimeter information are discussed later in this manual.

At the moment the SI-W and Scintillator-Fe systems are selected by default. The older implementations are still available within Brahms, but are no longer maintained. Details about the older calorimeter simulation are available in older versions of this note. Its use is discouraged, since the system is no longer supported.

2.8 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 [?]. In version Brahms_305 the geometry of the detector has been slightly changed to correspond to the detector presented by Chris Damerell at Hamburg in November 1998, and updated in spring of 2001. 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. A first estimate of material for cables etc in the foward region is included 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`.

2.9 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.

2.10 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`.

2.11 SI intermediate Tracker - SIT

The SIT has been implemented as a two layer SI detector in the barrel part. It is planned to use SI-strips for this detector. At this stage the simulation is rather crude. It does not include the effects of mirror and fake hits, and probably underestimates the amount of material needed to support these detectors.

2.12 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.

Access to the hits, tracks and results from the fits can be done either through a number of utility routines (see the TK package in the tracker part of the code for a complete list of these routines) or directly through statement functions, defined to map into the ZEBRA based storage of the hits and tracks. In the following the statement functions are documented.

```
*************************************************************************
* 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/
```

2.13 Track Reconstruction

A full pattern recognition module is available in Brahms which has been developed based on code from DELPHI and OPAL. Pattern recognition is performed in each subdetector capabale of doing so: SI detector, TPC, FCH at the moment. In a second step the track information from each individual sub-detector is merged using a merging processor. The structure and performance of the pattern recognition package is described in detail in [**?**].

Alternatively the user can select a "fake" option where no pattern recognition is done, but the MC information is used to define tracks.

In each case the tracks are then fitted using a KALMAN filter algorithm.

The track fitting 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.

2.14 Luminosity Calorimeter

The Luminosity Calorimeter and the Large Angle Tagger are implemented as described in the TDR as SI-W calorimeters with fine sampling in both the transverse and the longitudinal direction. The LAT is part of the calorimeter system and as such is used in the reconstruction of the energy flow in the event.

2.15 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.

2.16 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. The iron has been segmented according to the description given in the TDR. The information is available but at this stage has not yet been implemented in the reconstruction routines.

3 The HITS storage

Since release 308 Brahms supports the new common data model and storage model LCIO. Output of hits is realised in the LCIO data model. The reader is referred to the LCIO manual for a detailed documentation of the quantities stored, and for methods to access the information.

The hits file contains also information about the detector geometry used in the simulation. This is read by the reconstruction program an used to build it accordingly.

For backwards compatibility the old method of storing hits etc in Brahms has been retained. The following documentation is only concerned with the old method.

3.1 The old (pre LCIO) HITS storage model

Brahms offers the user the possibility to store the GEANT hits in a file and to read them back 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 binary stream with on-the-fly compression. A number of simple interface routines are provided to facilitate the writing and reading from the files.

As of Brahms version 306 no knowledge about the length of individual records is needed any more. The interface routines deduce this from the file and read in the appropriate number of bytes automatically. As a consequence the internal format of the hit files has changed between version 305 and 306. While Brahms 306 is backwards compatible and can read old-style files, only new styles ones can be written.

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
       (22)          IO version number
       (23)          code number to identify that version numbers
       (23) - #(4)   not used
-----------------------------------------------------------------------
MC information: a copy of the kinfile record
       (1) number of HEP-evt records
       (1)* 15 words per record

------------------------------------------------------------------------

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
                        |_______________________________________________
                        | 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 Storing Energy Flow Objects

The result of the processing of tracks, calorimeter information and muon system information are energy flow objects. The information about the energy flow objects is stored in a ZEBRA structure, located in the main store, with the top bank called **BDST**.

For each energy flow object a sub-bank is created, hanging at link $-neflow$, where $neflow$ is the number of the energy flow. The bank structure is shown in in Figure **??**.

The information for each energy flow object is stored in banks **BDS1** to **B999** (at the moment the number of eflow objects is limited to 300 by the SNARK program.). The energy flow objects are accessible through the statment function **REF(index, ieflow)** and **IEF(index, ieflow)** respectively. They are defined in the common **BRDFUNC2** and **BRDFUNC**.

For each energy flow object the following words are stored:

REF(1,i)	Px
REF(2,i)	Py
REF(3,i)	pz
REF(4,i)	E
IEF(5,i)	ID (only for MC)
REF(6,i)	1.0: object is linked to track
	2.0: object islinked to track, but track is secondary
IEF(7,i)	track number (TK) for charged, cluster number for neutral object
REF(8,i)	charge

There are a total of $nef(1)$ energy flow objects available. An additional bank is available at position $nef(1) + 1$, which collects the information on any hits in the calorimeter which are not connected with an energy flow object.

Each bank supports two banks: **BEHT** and **BHHT**, in which pointers at the hits in the ECAL (BEHT) and HCAL (BHHT) are stored. These pointers are accessible through the

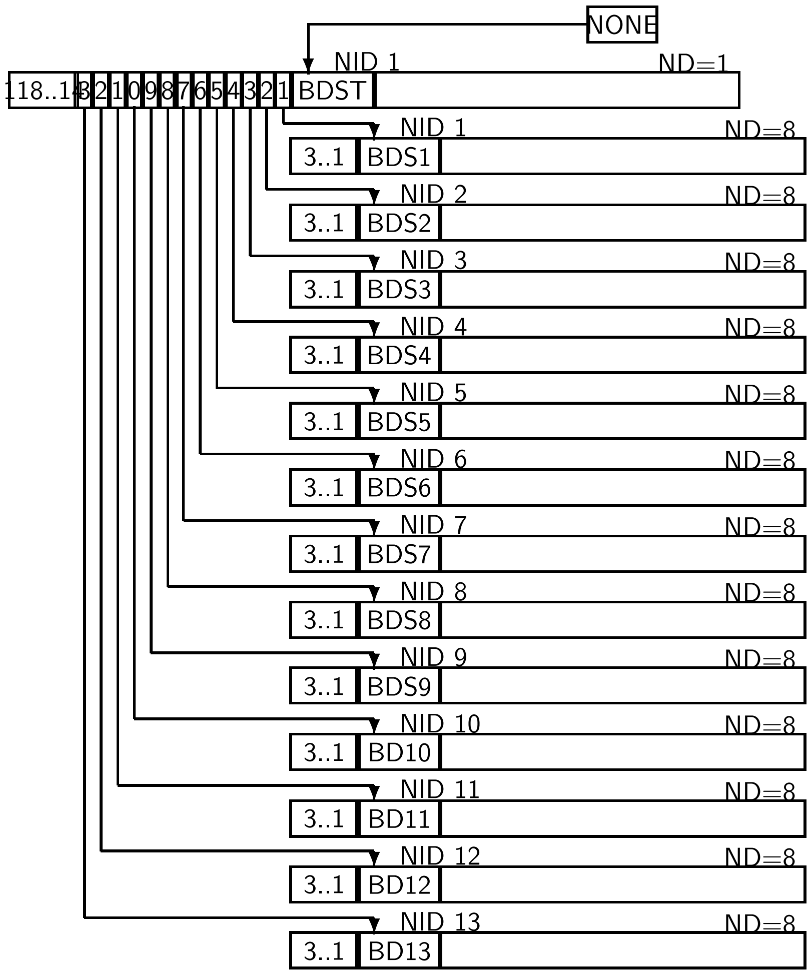

Figure 3: Typical bank layout for the energy flow objects.

statement functions: **IEF_EP(index, ieflow)** and **IEF_HP(index, ieflow)**. The pointers point straight into the common **ECRECONST** and **HBRECONST**, where the calorimeter hits are stored.

The word at position 7 of the energy flow object can be used to access the tracker information or the cluster information, depending on whether the objects is charged or neutral.

5 DST Output

It is our aim to implement the output of the reconstructed objects in LCIO as well if the LCIO data model is completed. Details will be described in the LCIO manual.

In addition Brahms is capable of writing an output rather similar to the one produced by SIMDET. Thus it should be possible to run the same analyses on SINDET and on Brahms output. The relevant part from the SIMDET manual is reproduced here - the details will probably change as the format of the records change over time.

The number of energy flow objects, **NEFLOW**, then follows. For each object, blocks of records follow, describing status information, best estimates, generator information, charged particles, calorimeter clusters and muons from the muon system, in this order.

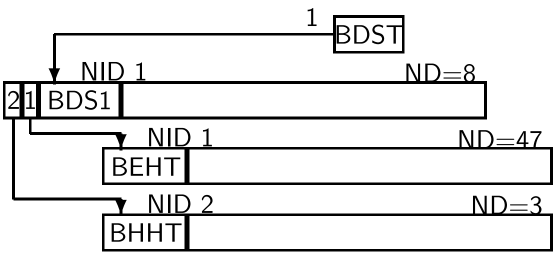

Figure 4: Layout of the pointer bank, supported by an energy flow bank.

File header record:

Offset	Variable name	Data type	Meaning
1	INSTATE	INTEGER	ISR flag, PYTHIA default: 0 = no radiation, 1 = ISR & Beamstrahlung, 2 = ISR only, 3 = Beamstrahlung
2	SQRTS	REAL	nominal collider cms energy [GeV]
3	SIGMA	REAL	Cross section [fb], if provided
4	IDCLASS	INTEGER	Reaction identifier, user defined
5	NEVENT	INTEGER	Number of events to be processed
6	IFBEST	INTEGER	Flag for best estimates
7	IFFORM	INTEGER	Flag for formatted output
8	IFDEDX	INTEGER	Flag for dEdx
9	IFHIST	INTEGER	Flag for complete (restricted) event history
10	IFBKGR	INTEGER	Flag for background events

This record is written once per file.

Generated particle record:

Offset	Variable name	Data type	Meaning
1	STATUS	INTEGER	0 = no detector response, 1 = detector response
2	ID	INTEGER	particle code according to PYTHIA convention
3	line	INTEGER	line number of particle in the PYTHIA record
4	Px	REAL	x-component of momentum [GeV]
5	Py	REAL	y-component of momentum [GeV]
6	Pz	REAL	z-component of momentum [GeV]
7	E	REAL	Energy [GeV]
8	m	REAL	mass of particle [GeV]
9	Q	REAL	charge of particle
10	x	REAL	x-component of vertex [mm]
11	y	REAL	y-component of vertex [mm]
12	z	REAL	z-component of vertex [mm]
13	Time	REAL	time of production [mm/c]

This record is repeated **NGENPA** times.

Energy flow record 1: status of energy flow object

Offset	Variable name	Data type	Meaning
1	STATUS	INTEGER	< 0: invalid object, 1=charged object, 2=neutral object, 3=composite object
2	Type	INTEGER	particle code according to PYTHIA convention, 999 = cluster of unresolvable neutral and charged particles
3	NGEN	INTEGER	number of generator particles contributing to the energy flow object
4	NTRK	INTEGER	number of charged particles contributing to the energy flow object
5	NCAL	INTEGER	number of clusters contributing to the energy flow object
6	NMUS	INTEGER	number of muons contributing to the energy flow object

This record and the following blocks are repeated **NEFLOW** times.

Energy flow record 2: best estimate for object's energy and direction

Offset	Variable name	Data type	Meaning
1	Px	REAL	x-component of momentum [GeV]
2	Py	REAL	y-component of momentum [GeV]
3	Pz	REAL	z-component of momentum [GeV]
4	E	REAL	Energy [GeV]
5	m	REAL	mass of particle [GeV]
6	Q	REAL	charge of particle

This record is written once per energy flow object.

Energy flow record 3: generator particle contributing to energy flow object

Offset	Variable name	Data type	Meaning
1	link	INTEGER	line number of particle in the PYTHIA record
2	Efrac	REAL	energy fraction

This record is repeated **NGEN** times.

Energy flow record 4: charged particle tracks that are part of the object

Offset	Variable name	Data type	Meaning
1	P	REAL	absolute value of momentum [GeV]
2	Theta	REAL	polar angle [radian]
3	Phi	REAL	azimuth angle [radian]
4	Q	REAL	charge of particle
5	Imp(R,phi)	REAL	transverse impact parameter [cm]
6	Imp(R,z)	REAL	longitudinal impact parameter [cm]
7	cov. matrix	REAL	xy-xy element
8	cov. matrix	REAL	xy-z element
9	cov. matrix	REAL	z-z element
10	cov. matrix	REAL	xy-theta element
11	cov. matrix	REAL	z-theta element
12	cov. matrix	REAL	theta-theta element
13	cov. matrix	REAL	xy-phi element
14	cov. matrix	REAL	z-phi element
15	cov. matrix	REAL	phi-theta element
16	cov. matrix	REAL	phi-phi element
17	cov. matrix	REAL	xy-1/p element
18	cov. matrix	REAL	z-1/p element
19	cov. matrix	REAL	theta-1/p element
20	cov. matrix	REAL	phi-1/p element
21	cov. matrix	REAL	1/p-1/p element
22	dEdx	REAL	signed electron probability
23	dEdx	REAL	signed muon probability
24	dEdx	REAL	signed pion probability
25	dEdx	REAL	signed kaon probability
26	dEdx	REAL	signed proton probability
27	dEdx	REAL	normalised electron dEdx value
28	dEdx	REAL	normalised muon dEdx value
29	dEdx	REAL	normalised pion dEdx value
30	dEdx	REAL	normalised kaon dEdx value
31	dEdx	REAL	normalised proton dEdx value

This record is repeated **NTRK** times.

Energy flow record 5: calorimeter cluster (ECAL/HCAL/LAT/LCAL) that are part of the object

Offset	Variable name	Data type	Meaning
1	E	REAL	energy [GeV]
2	Theta	REAL	polar angle [radian]
3	Phi	REAL	azimuth angle [radian]
4	Time	REAL	timing information
5	C(em)	REAL	probability of being consistent with an electromagnetic particle
6	E	REAL	energy [GeV]
7	Theta	REAL	polar angle [radian]
8	Phi	REAL	azimuth angle [radian]
9	Time	REAL	timing information
10	C(mip)	REAL	probability of being consistent with a minimum ionizing particle

This record is repeated **NCAL** times.

Energy flow record 6: muons from the muon system that are part of the object

Offset	Variable name	Data type	Meaning
1	E	REAL	energy [GeV]
2	Theta	REAL	polar angle [radian]
3	Phi	REAL	azimuth angle [radian]
4	Q	REAL	charge of particle
5	Time	REAL	timing information
6	C(punch)	REAL	probability of being consistent with an punch-through object

This record is repeated **NMUS** times.

6 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] The TESLA Technical Design Report, Part IV: A Detector for TESLA, Editors T.Behnke, S.Bertollucci, R.-D.Heuer, R.Settles, DESY 2001-011

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

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

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

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

[6] Kristian Harder etal. LCnote

[7] Reference to the DELPHI Kalman filter fit program

[8] SNARK: A reconstruction program for the highly granular TESLA calorimeter with analogue HCAL , by V. Morgnuov, Feb. 2002

7 Appendix: The Tracking Package

Kristian Harder

Track reconstruction in all Brahms versions since 2.00 is performed by a complex software package, the largest part of which has been adapted from corresponding software in actual use at the LEP experiments. Code from ALEPH, DELPHI and OPAL was merged with dedicated new code to achieve the desired functionality. This implies, however, that the Brahms track reconstruction package is a collection of completely different programming styles, making it sometimes hard to understand the structure and basic ideas of the code. This document intends to support further Brahms track reconstruction development, tuning and debugging by giving a walk-through through all relevant components of the track reconstruction software. Important parameters are discussed as well as known shortcomings of the code (as of Brahms 3.05).

First, a brief overview over the modules of the reconstruction code will be given. The main part of this document will then describe the track reconstruction process routine by routine. This document will conclude with a set of receipts for tuning the performance of Brahms and adding or changing subdetectors.

This note is based on Brahms version 3.05 as of November 1, 2002. However, most statements should be valid for a large variety of Brahms versions since 2.00.

7.1 Overview of the track reconstruction modules and data formats

Each package introduced in this section is merged into Brahms as individual program source file. The package names being referred to in the following correspond to the respective file name (except version number and file name extension) in the Brahms code subdirectory src/tracker/.

The steering package TKSTEER provides interfaces to all individual tracking packages and handles data storage, format conversion, user interface and global performance analysis. It runs a three-stage track reconstruction process: first, local track search in the individual subdetectors are performed. Three local track searches are implemented: SIPATREC investigates the silicon based inner tracking system of vertex detector (VTX), silicon intermediate tracker SIT and forward tracking disks FTD. TPCPATREC performs a track search in the TPC. FCHPATREC finally looks for tracks in the additional forward chambers behind the TPC end caps. TKSTEER provides hits in the tracking system in a common format, shown in Table **??** for easy access by local track searches. TKSTEER requires all local track search packages to present their output in the TE (track element) data format given in Table **??**. The TE format is general enough to accommodate both local tracks and single hits. This is used for the silicon pattern recognition, where all hits are passed on to later reconstruction steps along with the tracks found locally. This is done to allow a refinement of the local silicon track search, which suffers most of all local track searches from background hits, at a later stage of the global track reconstruction process.

The TE format is a mixed type (integer and real) array, which does seem a bit inconvenient and inflexible from today's standard, but it has to be used to avoid major rewriting of the second stage of the track reconstruction, coded in the DELSEARCH package. DELSEARCH attempts to find matching TEs from individual subdetectors. This is done by creating seed tracks out of TEs, extrapolating their parameters to all subdetectors in question, and picking up matching TEs from these subdetectors. In order to be able to pick up individual silicon hits, all silicon layers are treated as separate subdetectors in this code. DELSEARCH assigns TEs to any global track candidate where they fit into according to the track parameters and errors. It does not care about possibly ambiguous assignments of single TEs to more than one such global track candidates. The result of DELSEARCH is therefore not considered final, but is given in terms of TSs (track segments, see Table **??**) which have to processed further.

The third step of track reconstruction tries to reduce the set of partially redundant and ambiguous TSs to a consistent set of tracks (TKs, see Table **??**). This set is optimized by a maximizing a score function taking into account the total number of tracks, the number of TEs used in these tracks (with different weights for different subdetectors) and the track fit χ^2.

index	type	meaning	unit
1	real	x position	cm
2	real	y position	cm
3	real	z position	cm
4	real	deposited energy	GeV
5	integer	subdetector ID (Brahms convention, i.e. > 100)	—
6	integer	Monte Carlo track ID, 0 for real data (sic!)	—
7	integer	pointer to first exclusion (0 if none)	—
8	integer	number of exclusion list entries for this hit	—
9	integer	resolution code: bit 0 set if error on $r\phi$ given, bit 1 set for error on z, bit 2 for error on r. exactly two bits should be set.	—
10	real	first resolution word, corresponding to lower active bit at index 9	cm
11	real	second resolution word, corresponding to upper active bit at index 9	cm

Table 1: Data format for hit storage in the tracking banks. The mixture of real and integer variables in one array is achieved by declaration of a real and an integer array and equivalencing them. It is the responsibility of the user to know which index to access as which datatype. Indices 7 and 8 are used for mutually exclusive hit candidates (e.g. mirror hits in strip detectors). They point to entries in a separate exclusion list (array IEXCL in the TKBANK sequence).

The optimization is done in a recursive algorithm, for which the C programming language was preferred over Fortran. This code package consists therefore of two files: DELSOLVE contains the main C coded algorithm, and DELAMBI provides a Fortran-to-C interface for it.

All track reconstruction packages use the package DELFIT to perform track fits. DELFIT is a fast track fit package using a set of infinitely thin material planes as detector representation. These material planes are being set up in Brahms together with the detector geometry definitions. Deviations from detector setup and fit material database can lead to significant degradation of the tracking performance.

All packages described so far provide realistic track reconstruction algorithms, i.e. do not use any specific Monte Carlo information. However, for debugging and/or benchmark purposes, the full chain can be switched to ideal track reconstruction using FFREAD cards. Ideal (or "dummy") track search in the individual subdetectors is done in an extra package (DUMPATREC) to keep the complicated realistic track search packages free from extra code for this option. DELSEARCH does not require too much extra code for ideal subdetector merging; therefore this option is included in the main code itself. Since ideal merging does not lead to ambiguities, the DELSOLVE stage is per definition inactive in ideal reconstruction mode, and no extra measures had to be taken to accommodate for both reconstruction variants. Ideal reconstruction, as well as performance analysis of the realistic reconstruction, requires access to the Monte Carlo information. For this purpose, TKSTEER keeps a record of all Monte Carlo tracks (data format as in Table **??**).

index	type	meaning	unit
1	integer	subdetector ID	—
2	integer	submodule: bit code for VTX/FTD layers	—
		used in TE, ignored otherwise	
3	integer	not used (yet) - MUST be set to zero	—
4	integer	measurement code (see below)	—
bit	0	0 for x, y, z coordinates; 1 for R, $R\Phi$, z coordinates	
	1	always 0	
	2	1 if coordinate 1 is known (index 10 in TE), 0 otherwise	
	3	1 if coordinate 2 is known (index 11 in TE), 0 otherwise	
	4	1 if coordinate 3 is known (index 12 in TE), 0 otherwise	
		('known' means either defined by detector geometry or meas.)	
	5	1 if $R\Phi$ (x) is measured for bit 0 = 1 (0)	
	6	1 if z (y) is measured for bit 0 = 1 (0)	
	7	1 if Θ is measured	
	8	1 if ϕ is measured	
	9	1 if $1/p$ is measured	
	10	1 if $1/p_t$ is measured	
		('measured' really means measured, contrib. to the cov. matrix)	
	11	always 0	
	12	always 0	
	13	1 if $\mathrm{d}E/\mathrm{d}x$ is measured	
	14-...	always 0	
5	integer	pointer to end of TE (m+17, m=length of cov. matrix)	—
6	integer	charge (0=neutral, 1=positive, 2=negative, 3=unknown)	—
7	integer	number of degrees of freedom	—
8	real	χ^2 of the fit (if any — set to 1 otherwise)	—
9	real	length of the track element	cm
10	real	coordinate 1 of reference point: x or r	cm
11	real	coordinate 2 of reference point: y or $r\phi$	cm
12	real	coordinate 3 of reference point: z	cm
13	real	polar angle Θ at reference point	$0..\pi$
14	real	azimuthal angle ϕ at reference point	$0..2\pi$
15	real	$1/p$ or $1/p_t$ at the reference point (see meas. code)	$(\mathrm{GeV/c})^{-1}$
16	real	$\mathrm{d}E/\mathrm{d}x$ (ignored if not measured)	???
17	real	covariance matrix for measured quantities:	
		$R\Phi$, z (reference point at fixed radius R),	
		Θ, ϕ, $1/p$ has the following entries:	
		($R\Phi$,$R\Phi$),	
...		($R\Phi$,z), (z,z),	
		($R\Phi$,Θ), (z,Θ), (Θ,Θ),	
		($R\Phi$,ϕ), (z,ϕ), (Θ,ϕ), (ϕ,ϕ),	
		($R\Phi$,$1/p$), (z,$1/p$), (Θ,$1/p$), (ϕ,$1/p$), ($1/p$,$1/p$)	
m+16	real	error on $\mathrm{d}E/\mathrm{d}x$ if $\mathrm{d}E/\mathrm{d}x$ is measured	???
m+17	real	must be 0.0	—

Table 2: TE (track element) data format (from the DELPHI TANAGRA database system) for storage of subdetector contributions, both single hits and locally fitted tracks, to the global track search. The covariance matrix describes all parameters that are measured according to the TE measurement code. In the measurement code bits 5–10 describe the variables contributing to the TE covariance matrix. The order of the variables in the covariance matrix is the same as herein.

index	type	meaning	unit
1	integer	module code (internal use only)	
2	integer	bit code for used detectors	
3	integer	measurement code (see Table **??**)	
4	integer	track type: always 9 in Brahms	
5	integer	number of TEs used in this TS	
6	integer	charge (0=neutral, 1=positive, 2=negative, 3=unknown)	
7	integer	always 0	
8	integer	number of degrees of freedom of the track fit	
9	real	χ^2 of the track fit	
10	real	track segment length	cm
11	real	x of TS start point	cm
12	real	y of TS start point	cm
13	real	z of TS start point	cm
14	real	x of TS end point	cm
15	real	y of TS end point	cm
16	real	z of TS end point	cm
17	real	coordinate 1 of reference point: x or R	cm
18	real	coordinate 2 of reference point: y or $R\Phi$	cm
19	real	coordinate 3 of reference point: z	cm
20	real	Θ angle at reference point	$0..\pi$
21	real	ϕ angle at reference point	$0..2\pi$
22	real	$1/p$ at reference point (signed with geometric curvature of track measured from inside to outside of detector)	$(\mathrm{GeV/c})^{-1}$
23	real	covariance matrix for measured quantities (as above)	
...			
37	real		

Table 3: Data format for ambiguous global track candidates (track segments, TS). These represent the merging result as created by the TKSTEER package and DELSEARCH. Ambiguous use of TEs is possible here, e.g. use of the same TE in more than one TS; the unambiguous optimal track reconstruction is stored in TK banks (see Table **??**).

bit	meaning
0	0 no estimate at all for the TS parameters 1 TS parameters are given
1	0 for x, y, z coordinates 1 for R, $R\Phi$, z coordinates
2	0 crude estimate of error (diagonal matrix) 1 full error matrix available
3	0 all parameters are given if bit 0 is on 1 no error matrix, start point, end point

Table 4: Description of the measurement bit code used in the TS data format.

index	type	meaning	unit
1	integer	module identifier (internal use only)	
2	integer	bit code for used detectors	
3	integer	0 for x, y, z coordinates, 1 for R,$R\Phi$, z	
4	integer	track type: always 9 in Brahms	
5	integer	number of TEs used in this TK	
6	integer	charge (0=neutral, 1=positive, 2=negative, 3=unknown)	
7	integer	always 0	
8	integer	number of degrees of freedom of the track fit	
9	real	χ^2 of the track fit	
10	real	track length	cm
11	real	x of TK start point	cm
12	real	y of TK start point	cm
13	real	z of TK start point	cm
14	real	x of TK end point	cm
15	real	y of TK end point	cm
16	real	z of TK end point	cm
17	real	coordinate 1 of reference point: x or R	cm
18	real	coordinate 2 of reference point: y or $R\Phi$	cm
19	real	coordinate 3 of reference point: z	cm
20	real	Θ angle at reference point	$0..\pi$
21	real	ϕ angle at reference point	$0..2\pi$
22	real	$1/p$ at the reference point (signed with geometric curvature of track measured from inside to outside of detector)	$(\mathrm{GeV/c})^{-1}$
23 ... 37	real real	covariance matrix for measured quantities (as above)	

Table 5: Data format for final track candidates (TK). This information is stored in the arrays RTK/ITK of the TKBANK sequence in TKSTEER. The number of TK is stored in NTK. Whenever possible, this information should not be accessed using these variables directly, but through the TKSTEER interface routines TKREAD and TKNUMB.

index	type	meaning	unit
1	real	p_x at origin	GeV/c
2	real	p_y at origin	GeV/c
3	real	p_z at origin	GeV/c
4	real	E at origin	GeV
5	real	x of track origin	cm
6	real	y of track origin	cm
7	real	z of track origin	cm
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)	—

Table 6: Data format for storage of true (Monte Carlo) track information. This information is stored in ZEBRA banks and can be accessed via the statement functions TKMCTR/ITKMCT. The number of tracks is stored in TKNTRK. However, this information should usually be accessed by the TKSTEER utility routines TKREAD and TKNUMB, to allow for possible future changes of the internal implementation.

7.2 FFREAD cards

7.3 Routine-by-routine walk-through through the track reconstruction

7.3.1 TKSTEER - main program flow

initialization: TKINIT, TKSETR

Initialization of the track reconstruction code is done centrally by TKINIT and TKSETR in TKSTEER. TKINIT defines FFREAD cards and call initialization routines of individual tracking packages where necessary. TKSETR should be called after values have been read for the FFREAD cards. It verifies the usefulness of the user-supplied values for FFREAD cards and does FFREAD-card dependent initialization of other tracking packages. A large fraction of the TKSETR code is then dedicated to setting up the expected amount of noise hits in individual subdetectors and subdetector layers.

overall steering: TKTREV and supporting routines

The routines involved in the track reconstruction are show in figure **??**.

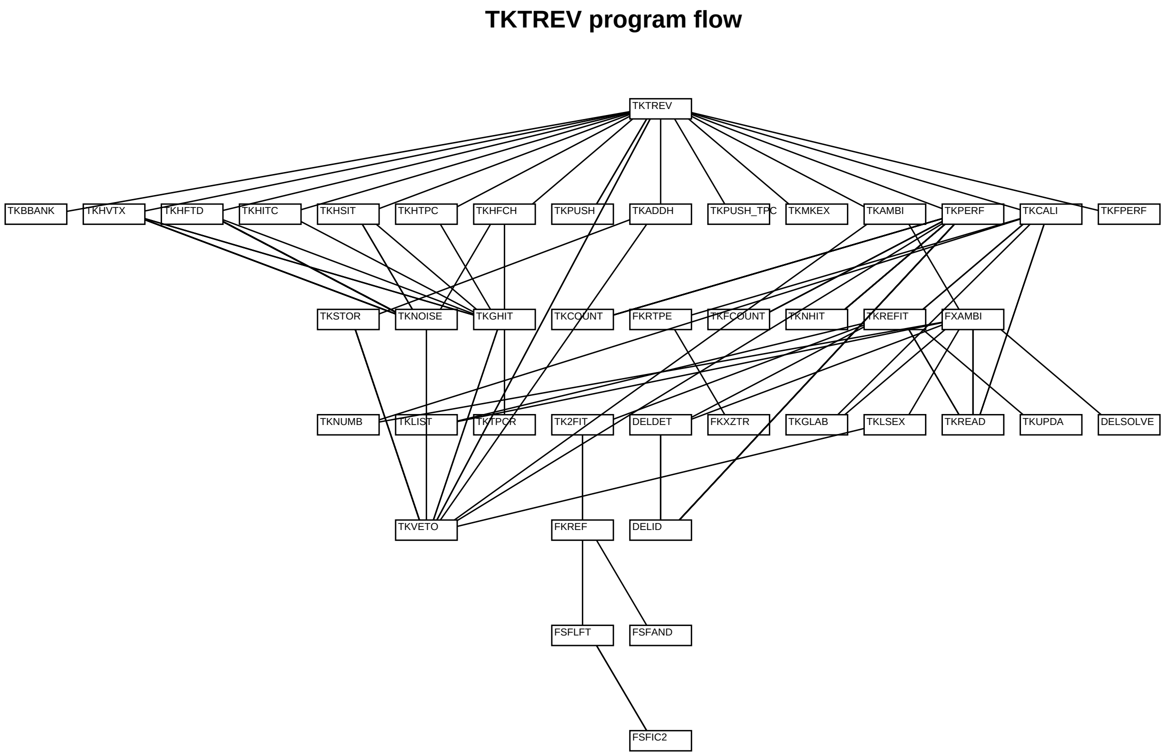

Figure 5: Flow chart for the track reconstruction.

The starting point for the track reconstruction is the routine TKTREV in TKSTEER. It is called from the overall reconstruction steering routine RERECO (and was called from GUTREV in Brahms 2.xx, hence the name). This routine begins with the event initialization, i.e. initialization of the ZEBRA banks for tracking (via TKBBANK) and reset of all pointers and index variables. It then calls routines that transfer all GEANT hits to the tracking hit store, where they can be easily (and independently of the GEANT version) accessed by the local track searches. Other than the GEANT banks, the TK hit bank contains also information on the precision of the hit coordinates, which is subdetector-specific. Therefore it is done separately by a dedicated routine for each subdetector: TKHVTX (vertex detector, CCD or APS), TKHFTD (forward tracking disks), TKHSIT (silicon intermediate tracker), TKHTPC (TPC), TKHFCH (forward chambers). These routines are also responsible for injecting the number of noise hits defined in TKSETR. They are mere interfaces to a generic noise hit injection routine (TKNOISE) and a generic hit readout routine (TKGHIT).

The track reconstruction itself begins with a sequence of calls of local track search routines. Depending on the FFREAD card setup, either ideal pattern recognition routines ("LP_detname") from DUMPATREC are called, or realistic pattern recognition routines from the respective packages. The realistic silicon track search requires a second routine to be called after the track search itself: it was mentioned earlier that silicon hits are to be passed on to later stages of track reconstruction to allow a later review of the assignment of hits to tracks. However, the SIPATREC package only returns tracks, not the hits which have not been assigned to a track. This is done by TKADDH instead. It should be noted that the track reconstruction routine call statements are surrounded by code which records the time required by the individual packages, allowing a final CPU consumption statistics at the end of a run. After all TEs from the subdetector track searches have been found, a call to TKPUSH adjusts the ZEBRA bank size accordingly, so that all remaining space is available for subsequent steps of the event reconstruction.

The next part of TKTREV is dedicated to book-keeping mutually exclusive TEs. Two TEs are considered mutually exclusive if they either share hits or if hits in one TE are mutually exclusive with hits from the other TE. The former is simply a means to describe that the track reconstruction considers it impossible that a single hit can be part of more than one track. The latter, hit exclusions, comes from the method used to treat strip detectors in the track reconstruction. Hit reconstruction in strip detectors leads to ambiguous hit candidates: true hits and their mirror hits. It is not a priori which hits are true and which are mirrors, but it is clear that out of a given set of a true hit and its mirror hits, only one hit candidate can be used in the final track candidates,whereas the others have to be discarded at some point. This is ensured by marking each set of true and mirror hits as mutually exclusive. Which hit will actually be chosen as true is left to the final (DELSOLVE) stage of track reconstruction, where the best track fits are selected out of all possible combinations suggested by DELSEARCH. To emphasize this, at this stage in TKTREV, exclusions are just registered in a list during a call to TKMKEX; no attempt is taken at this point to resolve any exclusions.

The global track search defines the next section of TKTREV: a call to FSFSTR (in DELSEARCH) runs the subdetector merging processor which creates TSs. A subsequent call to TKAMBI starts the final ambiguity resolver. TKAMBI is the interface to the interface to the C code in DELSOLVE — a slightly unfortunate construction arising from the idea not to touch the original DELPHI Fortran-to-C interface code in DELAMBI too much. Sole purpose of TKAMBI is to copy all TSs into the TK bank, since DELSOLVE operates on the TK bank exclusively. After running DELSOLVE, TKAMBI also takes care of wiping track candidates that were marked as "not to be used" by DELSOLVE from the TK bank.

After the track reconstruction itself was done, a call to TKPERF (if selected by FFREAD cards) evaluates the performance of the reconstruction in this specific event. Finally, a call to TKCALI interfaces the track reconstruction result to the calorimeter code. TKCALI has become a major piece of code due to completely different data structures in tracking and calorimetry. First of all, TKCALI attempts to extrapolate all track parameters to the inner face of the calorimeter. This is done by using a feature of the DELFIT package: all surfaces defined in the common blocks FKEXTS and FKEXTY (both in patchy keep sequence FKEXTS) are used as target extrapolation surfaces, and the track fit will assign a set of track parameters to all surfaces that are being traversed by the fitted track. TKCALI replaces the regular set of extrapolation surfaces temporarily by just three ones, one cylinder and two end cap planes, which outline the outer boundary of the tracking detectors. Then all tracks are refitted with the help of TKREFIT, which dissects every TK into its individual hits, forms TEs out of those hits and refits the track. Following this refit, the track parameters outside the tracking detector volume will be available as extrapolated track parameters on one out of the three extrapolation surfaces. In the case of the end cap planes, this is exactly at the inner boundary of the calorimeter. Tracks ending up in the barrel do however require another step due to the octagonal shape of the calorimeter. Octagonal extrapolation surfaces, as well as planar surfaces other than those orthogonal to the beam axis, are not foreseen in DELFIT. Therefore a second barrel extrapolation surface is defined for each barrel track. The radius of this barrel is chosen such that it coincides with the inner calorimeter radius at the ϕ coordinate where the track leaves the tracking detectors. This is an approximation which is good for tracks of large momentum, but becomes increasingly imprecise for small momentum tracks.

Along with the track parameters at the outer end of each track, TKCALI calculates the Perigee parameters (especially distances of closest approach to the interaction point in xy and z, d_0 and z_0). Both parameter sets are stored in the CALOINPUT keep sequence (see Table **??**).

7.4 How to...

The intention of this section is to provide Brahms developers with all information necessary to perform specific tasks involving major changes to the tracking code.

7.4.1 How to add a new detector

The addition of a new tracking detector requires only a rather small number of steps. The tracking code is quite generic, so the structure of the code should hardly be affected at all. Whenever reference to a specific example helps to illustrate the necessary tasks, the new detector will be referred to as NEW (Non-Existing Wire chamber).

First of all, the geometry definition and declaration of a sensitive volume have to be added to the GEANT declarations in Brahms. A new detector ID, ID_NEW, has to be defined, plus possibly subcomponent (layer) IDs ID_NEW1, ID_NEW2 etc. The generic subdetector ID ID_NEW should be a multiple of 100 (Brahms detector ID convention). Subcomponent IDs can then take any useful value between $ID\_NEW + 1$ and $ID\_NEW + 99$. Details of how to create a detector in GEANT are not discussed here.

The track reconstruction system has to know what type of track reconstruction (ideal or realistic) it is supposed to use for this detector. This requires an entry in the *ipatrc* array. Increase the array size of *ipatrc* in the TKFFREAD keep sequence in TKSTEER by one. The FFKEY statement in TKINI has to changed accordingly, and the upper limit of the DO loop that sets the default values in the same routine has to be increased. In TKSETR, the detector name array *chdet* has to be extended by one entry with the name of the new detector. The increased number of detectors has to be taken care of in two DO loops in this routine.

The creation of GEANT hits for the new detector has to be steered by a routine similar to TPCDHIT et al. — just add a routine NEWDHIT to account for hits from this detector and make sure it is called from GUSTEP.

The next step is to provide means to read the hits from the new detector and make them available for the track reconstruction routines. A new subroutine TKHNEW is needed which uses the generic hit read routine TKGHIT to retrieve hits and store them in the tracking data arrays. The new routine should be modeled after TKHTPC etc. A call to this routine has to be added to TKTREV.

variable	type	meaning	unit
		MC information	
ninp0	integer	number of MC tree particles	—
lpart0(i)	integer	particle ID code for this particle	—
lvert0(i)	integer	Address of particle in HEP record	—
xmass0(i)	real	MC mass of particle	GeV
xcharge0(i)	real	charge of particle	e
xc0(i)	real	origin x of particle	cm
yc0(i)	real	origin y	cm
zc0(i)	real	origin z	cm
cx0(i)	real	p_x/p (true momentum)	—
cy0(i)	real	p_y/p	—
cz0(i)	real	p_z/p	—
pm0(i)	real	p	GeV/c
		charged track information at the point of closest approach to the IP	
ninp1	integer	number of charged tracks (equal to number of TK)	—
lpart1(i)	integer	particle ID (if MC)	—
lvert1(i)	integer	MC vertex number	—
xmass1(i)	real	mass of particle	GeV
xcharge1(i)	real	charge (-1., 0., 1., or 10. if unknown)	e
xc1(i)	real	$-d_0 \sin(\phi)$	cm
yc1(i)	real	$d_0 \cos(\phi)$	cm
zc1(i)	real	z_0	cm
pm1(i)	real	$\|p\|$	GeV/c
cx1(i)	real	$\sin(\Theta) \cos(\phi)$	—
cy1(i)	real	$\sin(\Theta) \sin(\phi)$	—
cz1(i)	real	$\cos(\Theta)$	—
		charged track information at the face of the calorimeter	
ninp2	integer	number of charged tracks (equals to number of TK banks)	—
lpart2(i)	integer	particle ID (if MC)	—
lvert2(i)	integer	MC vertex number	—
xmass2(i)	real	mass of particle	—
xcharge2(i)	real	charge (-1., 0., 1., or 10. if unknown)	—
xc2(i)	real	x coordinate at calorimeter face	cm
yc2(i)	real	y coordinate at calorimeter face	cm
zc2(i)	real	z coordinate at calorimeter dace	cm
pm2(i)	real	$\|p\|$ at calorimeter face	GeV/c
cx2(i)	real	$\sin(\Theta') \cos(\phi')$	—
cy2(i)	real	$\sin(\Theta') \sin(\phi')$	—
cz2(i)	real	$\cos(\Theta')$	—

Table 7: Track parameter format as used in the TKCALI output. This information is stored in the CALOINPUT sequence in BRCDES.

TKHNEW must also take care of adding an adequate amount of noise hits to the hit data banks. The implementation of noise requires a few additional steps. First of all, a new variable $TNNEW$ (or, for a multi-layer detector, variables $TNNEW1$, $TNNEW2$, ..., $TNNEWn$) has to be added to the TKNOISEPAR sequence in TKSTEER. Add initializations of this variable/these variables to TKSETR in TKSTEER, depending on beam energy ($TNENERGY$) and magnetic field ($BFIELD$). This is done in a huge IF-ELSE-statement.

During track reconstruction, the information provided by the new subdetector has to be presented in DELPHI TE format (see Table **??**). For detectors without local pattern recognition, this is easily achieved by using the generic dummy pattern recognition routine $\mathrm{LP_DET}$ in DUMPATREC. See $\mathrm{LP_FCH}$ for usage of this routine. For a detector without capability of local pattern recognition, $\mathrm{LP_DET}$ will just return all the hits in TE format. The routine $\mathrm{LP_NEW}$ calling $\mathrm{LP_DET}$ for the new detector must then be called from TKTREV. In order to call $\mathrm{LP_DET}$, one needs the Brahms detector ID of the new detector (or, respectively, all subcomponent IDs), the detector type (plane, cylinder, cone) and the DELPHI measurement code to be assigned to the TE - both for hits and possibly fitted tracks. See Table **??** for the definition of this code. LP_SI might give nice examples for most detector types. LP_DET furthermore requires a pattern recognition code. A value larger than zero asks LP_DET to store hits and tracks (if any). Values below zero prevent hit TEs from being created — only fitted tracks are returned. So far this option is only used for the TPC. Since no attempt is done to improve the TPC pattern recognition at a later stage of the track reconstruction, storage of all TPC hits is considered an unnecessary burden. Multi-layered detectors whose hits are to be used in the track reconstruction must have a separate detector ID for each layer, otherwise track reconstruction will simply not work: the DELPHI code takes at most one TE per detector ID to construct a track candidate. LP_NEW has to be called in TKTREV close to where the corresponding calls are for the existing subdetectors.

For a detector with local pattern recognition capabilities the action at this point should depend on the entry in the $IPATRC$ array associated with the detector. A value of 1 is used for realistic pattern recognition. This requires a custom routine which should read hits with the routine TKREAD and return collections of hits to be fitted as tracks using TKMKTE. A value of 2 in $IPATRC$ asks for ideal pattern recognition. If the code is implemented as described above for detectors without local pattern recognition, everything should work automatically. LP_DET fits tracks whenever possible, i.e. it attempts to fit tracks whenever one track leaves at least three hits in the respective subdetector. Depending on the value in $IPATREC$, either the custom realistic pattern recognition routine or LP_NEW have to be called in TKTREV close to where the corresponding calls are for the existing subdetectors.

At this stage, everything should be in place to provide the track merging processor DELSEARCH with the information it needs. However, DELSEARCH itself requires a couple of adaptations for the new detector. First of all, a detector number has to be assigned to the detector. This has to be done in the FSFPARAMS sequence in DELSEARCH. Add a new parameter definition for i_NEW to the existing list of parameters i_TPC, i_FCH etc. If the new detector is capable of local pattern recognition, one might want to start the merging process from this detector (among other strategies, of course). In this case, i_NEW has to be inserted early, before i_SCAN1. All detectors capable of functioning as seed for the track merger have to be listed in the beginning, and their total number *maxpd* (max primary detectors) has to be increased by one when adding the new detector there. For non-primary detectors i_NEW must be larger than all i_SCANn values, and *maxpd* stays the same. In any case, the total number of detectors, indicated by *maxdet*, must be increased by 1. Each submodule ID requires its own parameter, i.e. i_NEW1, i_NEW2 etc. if applicable. (Applicable means that the new detector has submodules and the submodule hit TEs are stored in the tracking banks for usage in the merging processor. See comments on LP_DET usage given earlier. E.g. there are no individual parameters for all TPC pad rows, because individual TPC hits are not passed on to the merging processor.)

DELSEARCH requires some more information on the new detector in the FSFPARAMS sequence. The variable *detorder* needs an entry for the new detector. It describes the printing order of detectors whenever track candidates are shown on the screen. *detname* needs an entry for the new detector to tell DELSEARCH what name to print for the new detector. The variable *detid* needs another detector ID for the new detector. Track candidates are described as bit patterns of contributing subdetectors in DELSEARCH. Therefore each detector (submodule, layer) needs to have a unique bit number between 0 and 31 assigned. The values of 0, 1, 2, 30 and 31 are reserved for internal purposes. The value of *detid*(i_TPC)=5 should never be changed.

The routine FSFINP in DELSEARCH is responsible for reading all subdetector TEs from the tracking banks. It needs to know the correspondence of Brahms ID ID_NEW (ID_NEW1, ...) and i_NEW. Add this to the long IF-ELSE-statement. The correspondence between Brahms ID and DELPHI ID (*detid*(i_NEW)) should be taken care of automatically in the routine DELDET in TKSTEER. In case the identification of subdetectors in DELSEARCH fails to work, this routine is a good place to check.

A diagnostic routine in DELSEARCH, FSFPRINTTE, also wants to know a detector name. It should be added to the IF statement in that routine.

The next issue is to tune the merging algorithm itself. FSFANA3 in DELSEARCH loops over all possible subdetector combination using a large number of nested DO loops. Find out what the maximum number of tracking subdetectors is a track can traverse in the detector. In standard Brahms, this is 5 CCD layers, SIT1, SIT2, TPC, FCH (TPC and FCH do not return individual hits, so there is only one TE for the full detector), i.e. 9 detectors — correspondingly more after adding another detector. FSFANA3 needs at least this number of nested DO loops to operate correctly. If necessary, add more loops, including the IF statement accompanying each loop header. Don't forget to update all variable names, goto destinations and statement labels when doing copy/paste to add more loops!

Technically, DELSEARCH is now able to treat the new detector correctly. However, with the above changes applied, it will still not be used in the merging process. DELSEARCH needs definitions of track search strategies that include the new detector. This is done in FSFSTR. There we have a sequence of track searches starting from various detectors (TPC, SI), and a few re-iterated searches starting from partially merged tracks (SCAN1, SCAN2 etc.). For each existing search one has to define whether the new detector should be included in this specific approach. $method(i\_NEW)$=0 does not attempt to merge any TE of the new detector into track candidates at this stage. $method(i\_NEW)$=$imethod$ means that DELSEARCH should attempt to extrapolate track candidates from the primary detector of the respective search to the new detector and attempt to find suitable TEs. Whether this merging is done according to matching track parameters (realistic mode) or according to matching Monte Carlo ID of the track elements (ideal track reconstruction) is determined from the value of the FFREAD card MCMERGE (true=ideal, false=realistic).

If the new detector is a primary detector, one will probably also want to start a track search from it. Copy for example the TPC track search section somewhere before the "cleanup and iterate searches" section in FSFSTR. Make sure that the new search defines the new detector as starting point of the track search by setting $method(i\_NEW) = none$. The first argument of the FSFANA3 call has to be adapted as well. Furthermore, to allow for time consumption analysis, change the argument of the *time* array references to i_NEW. One should also add a statement whether the TPC (or whatever detector was primary detector in the track search we copy-pasted from) should be included as target in our new track search: $method(i\_TPC) = method$ or 0. The call to FSFANA includes a bit code setting some flags, the definition of which can be found in Table **??**.

Further changes have to be applied to DELSOLVE. As DELSOLVE is C code as opposed to FORTRAN as used in DELSEARCH and TKSTEER, it uses its own set of symbolic constants to describe the subdetectors. Add the value of $detid(i\_NEW)$ to a definition of NEW_CODE in the list of compiler directives defining TPC_CODE, FTD_CODE and so on. The array $DetScore$ defines how important it is to keep a TE of a specific subdetector in a track candidate. If the importance is very low, one might tend to throw out this subdetector if this increases the track fit quality significantly. Add a value for the new detector to this array to reflect the relative importance of it. For example, standard Brahms assigns 20 points to a TPC track, whereas each SI hit gives 5 points. A value of -1 in the array marks an unused detector ID.

At this point Brahms should be able to reconstruct tracks including the new detector. Every TE of the new detector will be added to the track where it fits best — unless all track fits involving this specific TE fail. This means that all track combinations involving the new detector are tested. This might be rather slow, but should give the largest reconstruction efficiency that Brahms can achieve with this system. If that does not work, at least one item of the list of changes described in this section was not correctly implemented.

To speed up the merging without too much loss of efficiency, the DELSEARCH can preselect matching TEs looking for matching track parameters. Usage of this feature requires some tuning. The routine FSFINI in DELSEARCH defines cuts for the helix extrapolation of track

flag	meaning
FLAG_DATA	select TE merging according to measured hit/track parameters
FLAG_LABL	select TE merging according to MC track ID
FLAG_USEUSED	allow to add same TE to more than one track candidate
FLAG_GIVESINGLTE	allow single TE to be used as track candidate (e.g. TPC)
FLAG_PCONST	constrain momentum with beam spot
FLAG_SCANDET	scan previous track candidates for missing detectors
FLAG_CUT100	check at most 100 TEs per subdetector (layer)
FLAG_NOPLINV	do not use polar inversion to get initial track parameter estimate
FLAG_AVERAGETHETA	for polar inversion: use average Θ of all TEs instead of primary TE only
FLAG_BEAMSPOT	use beamspot constraint in track fit

Table 8: List of bit flags used to steer the behavior of the track search engine FSFANA3. Track parameter estimation by polar inversion is foreseen for angular regions without standalone pattern recognition capabilities. It should not be necessary in Brahms.

parameters from each subdetector to each other. The cuts define a maximum deviation of track parameters of the target detector TE compared to extrapolated values of the primary detector TE. (Where no cuts are defined, the default behavior is to combine all TEs that survive a common track fit, as described above.) Possible cuts are given in Table **??**. For tuning, add arbitrary cuts on all variables you consider useful. Cut values of 0 are recommended for all quantities for tuning, as tuning can be done with any cuts, and the harder the cuts are, the faster Brahms runs. Switch on DELSEARCH performance histogramming (FFREAD card DESHST 1) and run a few events. The number of events should be large enough to contain a total of a thousand tracks or more. After the run, look at the histograms in the DELSEARCH subdirectory of the tracking histogram output file. Two histograms are filled for each combination of detector i_FROM and detector i_TO if cuts on the extrapolated track parameters are defined in FSFINI. The histogram IDs for any such extrapolation are $10000 \times i\_FROM + 50 \times i\_TO + 10$ ... +19 for correct combinations, and +20 ... +19 for wrong combinations. All combinations are filled into the histogram, even if they fail to pass the cuts defined in DELSEARCH. It might be necessary to adjust the histogram range and/or binning towards the end of FSFINI in DELSEARCH. Optimized cut values can be found best by plotting the correct and the wrong combinations on top of each other with log scale on the y axis.

The overall performance of Brahms track reconstruction can be evaluated by switching the FFREAD card TKPERF to 1. Then, a file tkperf.summary will be created, showing a text summary of

- local pattern recognition performance
- TS contribution: how many tracks that have hits in a specific subdetector are reconstructed using the hits in this detector (after DELSEARCH)
- TK contribution: same as TS contribution, but for final track candidates (after DELSOLVE)

7.5 Acknowledgments

The Brahms track reconstruction code has been written, adapted from LEP code and debugged by many people, notably Heiko Bauke, Ties Behnke, Grahame Blair, Ivanka Božović-Jelisavčić,

variable	meaning
xcx	cut on agreement of x coordinates
xcy	cut on agreement of y coordinates
xcRPhiR	cut on agreement of $R\Phi$ coordinates
xcz	cut on agreement of z coordinates
xctheta	cut on agreement of Θ coordinates
xcphi	cut on agreement of ϕ coordinates
xcthest	cut on agreement of Θ^* directions
xcphist	cut on agreement of ϕ^* directions

Table 9: List of track parameter variables, cuts on which define compatible and incompatible TE parameters. A second set of variables, starting with *pl* instead of *xc*, is available for cuts on parameters obtained by polar inversion.

Markus Elsing, Jakob Hauschildt, Richard Hawkings, Klaus Mönig, Vassilly Morgunov, Harald Vogt, Daniel Wicke, Stefania Xella, not to mention the original authors of the LEP software, who certainly contributed the most significant fraction of the code.