ERHIC-SOFT/09-01
15 June 2009

ESIM

P.Nevski

Abstract

A simulation framework presented here includes a simulations program based on Geant3 and a built-in support for a flexible geometry description. It simplifies the description of detector geometries, automates to a large extenbt the detector response simulations, simplifies the digitisation coding and provides a data handling mechanism with a built-in documentation and database support.

1 Introduction

In the bcomming years the eRHIC collaboration have to made a number of detector choices on the basis of the detailed detector MC simulation. A fast and reliable way to implement these versions is to use a dedicated *geant* parser (Fortran preprocessor) which is supported by a special GEANT interface library. Maintaining the GEANT specific tables of materials, volumes, hits descriptions, etc and insuring the internal consistency of most of the actual parameters of the GEANT routines, it significantly reduces the amount of information the user should care of and improves the robustness of the program. Here we describe the main rules and features of this program.

2 GEOMETRY DESCRIPTION

The geometry of each detector in ESIM is described in a single module. Modules are written in the *geant* language and translated by the parser into conventional, well commented Fortran subroutines compiled and linked with the rest of detector deascription. A module consists of the module header, the data definition part and of a number of blocks, each describing one GEANT elementary volume and its content.

2.1 *geant* language

The *geant* language is a Fortran extension oriented to the GEANT application. Apart from standard Fortran statements, it contains a number of *geant* statements in the form:

$$\text{OPERATOR NAME [} keyword_1{=}value \ \ldots \ keyword_n{=}value \text{]}$$

where the OPERATOR defines a specific service to be performed by the *Atlsim* interface. Apart from the declarations and data handling operators, described in sections 2.4 and 2.5, there are 9 GEANT dedicated operators and 3 control operators in the *geant* language. For these operators:

- NAME is the name of a GEANT volume or of a volume shape (4 letters), or a material or medium name (up to 20 letters). A Fortran string variable is generated by the parser by converting the NAME into upper-case letters.
- Keywords (left parts of assignment) are variables used in the GEANT3 manual [1] to describe the parameters of the corresponding GEANT3 routine [1)].
- Their values (right parts of assignment) are any legal Fortran expression.

The language is neither case nor position sensitive. A *geant* statement can be continued on the next line only using a comma or an underscore at the end of a line as a continuation sign. A comma can also be used between keywords to improve the readability. All *geant* comments mentioned below are mandatory. They should not contain single or double quotas inside.

A list of keywords with their values is called below a definition.

2.2 Volume description

2.2.1 General structure

Any GEANT volume in a module is described as a block. A block consists of two parts - the description of its own properties and the description of its content - and has the following structure (last column shows the corresponding GEANT3 routine) :

1) Few deviations from this rule where the manual names are ambiguous will be mentioned later

	BLOCK NAME comment	
	MATERIAL name definition	⇒GSMATE
or few	COMPONENT name definition	⇓
followed by	MIXTURE name definition	⇒GSMIXT
	MEDIUM name definition	⇒GSTMED
	ATTRIBUTE name definition	⇒GSATT
	SHAPE name definition	⇒GSVOLU
		⇒GSDV(*)
a number of	CREATE name	
followed by	POSITION name definition	⇒GSPOS(P)
	HITS name definition	⇒GSDET(*)
	DIGI name definition	⇒GSDET(*)
	ENDBLOCK	

BLOCK, ENDBLOCK and CREATE are control operators because they affect the execution order: CREATE is executed as a jump to the requested BLOCK code and the return back when its ENDBLOCK is reached. All other are GEANT dedicated operators and are substituted by a call to one or few GEANT routines via the *Atlsim* interface.

Example 1:

```
Block     GAAS   is GaArsenid forward tracker
  Material   Air
  Medium     gaas_mother   Ifield=1,   FieldM=2,  TmaxFd=3,   Epsil=0.001,
                           SteMax=0.001   DeeMax=0.05   StMin=0.001
  Shape      TUBE          Rmin=10    Rmax=50   dz=200
  do idisc = 1,nint(gaag_Ndisc)
     Create     GDSi
     Position   GDSi   z=+gdsi_Zdisc
     Position   GDSi   z=-gdsi_Zdisc  ThetaZ=180
  enddo
endblock
```

The *Atlsim* interface maintains GEANT tables of materials, mediums, volumes and rotation matrices. After checking that the requested name already exists in the corresponding table or having created a new table entry, the interface provides the entry number to the GEANT routines.

The only mandatory operator inside a block is its SHAPE, others can be omitted. In this case the volume properties are inherited from it's mother volume, and position definitions are assumed to be default (x=y=z=0, no rotation).

If needed, material, medium and attribute operators should be defined before the SHAPE operator.

2.2.2 More on SHAPE

The *name* argument of the SHAPE operator contains a name of any of the 16 legal GEANT shapes described in the manual. Keywords in the definition part are the names of parameters, used in the GEANT manual (section GEOM 050) to describe these shapes. The only exception are multiple z, $Rmin$ and $Rmax$ parameters of the PCON and PGON shapes, which should be supplied as vectors named Zi, Rmn and Rmx, defined in one of the following two forms:

$$vector = \{val_1, val_2 ... val_n\}$$
$$\text{or } vector = \{A(i_1 : i_2)\}$$

where a *vector* stands for Zi, Rmn or Rmx, val_i are any Fortran expressions, and A is a Fortran array.

As the parameters are transmitted to the GSVOLU routine via the *Atlsim* interface, they can be provided in any order or be inherited from the mother volume.

Example 2: the PCON specification from the GEANT manual (GEOM 050, figure 23) may look like:

```
SHAPE PCON  phi1=180  dphi=270  Nz=4  Zi={-400,-300,300,400},
            Rmn={50,50,50,50}    Rmx={250,100,100,250}
```

The GEANT divisions are in the *geant* language particular cases of the SHAPE operator. The actual division mechanism is automatically selected by the *Atlsim* interface dependent on the parameters supplied.

Example 3: this will create divisions of a TUBE in ϕ using GSDVN (GEOM 130):

```
   ....
   Shape  TUBE       Rmin=Rj  Rmax=Rj+Dr  Dz=D/2
   Create GDij
   ....
Block     GDij    is a sector containing one counter
   Shape  DIVIsion   Iaxis=2  Ndiv=Ndv
   Create ....
endblock
```

2.2.3 Inheritance rules

Unless defined explicitly, parameters of the MATERIAL, MIXTURE, MEDIUM, and SHAPE operators in a new block are inherited from the block creating this one. Normally this is also its mother volume [2]. If no material or medium are defined in a new block at all, the are inherited from the mother block. A MATERIAL or a MEDIUM operator without parameters can be used to get parameters of already defined materials (mixtures) or media. If no material or medium are defined in a new block at all, the are inherited from the mother block.

A new GEANT medium, which combines both material and tracking parameters, is introduced not only after a MEDIUM operator re-defines any of the tracking parameters, but also if a material has been changed by a MATERIAL or MIXTURE operators.

A MATERIAL or a MEDIUM operator without parameters can be used to refer to an already defined material (mixture) or medium.

2.3 Volume positioning

Unless defined explicitly, the parameters of a POSITION operators have the default values:

$$x = y = z = 0, \text{ KONLY='ONLY}, \text{ unit rotation matrix.}$$

If the volume being positioned has been defined with all parameters equal to zero, the GSPOSP routine will be called, otherwise the GSPOS is used. In case of the GSPOSP call, the actual parameters of the volume shape supplied in the POSITION operator still follow the inheritance rules for the SHAPE operator.

If a rotation should be defined when positioning a volume, it is possible to define it in two ways:

– Providing up to 6 parameters of the GEANT rotation matrix (GEOM 200). The parameter names [3] and their default values defining the unit matrix are

$$ThetaX = 90^o, PhiX = 0^o, ThetaY = 90^o, PhiY = 90^o, ThetaZ = 0^o, PhiZ = 0^o$$

Only those parameters which are different from the default unit matrix should be given. Example: ThetaZ=180 in the example 1 in the second POSITION operator makes the second copy of the GDSi volume to be positioned as a mirror reflection of the first one.

– A rotation around one of the x, y, z axis can be introduced simply by defining one of the following parameters:

$$AlphaX, AlphaY \text{ or } AlphaZ.$$

Rotation parameters are not inherited from one POSITION operator to another.

2.4 Volume naming mechanism

All volumes in ESIM are referenced by their generic names, consisting of 4 upper-case letters [4]. When the real dimensions of the same generic volume are variable, the supporting *Atlsim* library provides an automatic and transparent mechanism which, for physically different volumes with the same generic name, generates nicknames used by GEANT, by changing last letters of the generic name into numbers or lower-case letters. These volumes with different nicknames are considered as instances of the same generic object. The original generic name is also kept in each instance together with its nickname.

The positioning of all volumes is done using their generic names, the latest generated instance of the object been actually used. When positioned in the same mother volume such instances will be made different also by their GEANT *copy numbers*. If a volume instance has been defined with all parameters equal to zero, it will be positioned by the *Atlsim* interface using the GSPOSP routine with the dimensions, defined in the POSITION operator.

[2] The exception is done only for the mentioned above vectors Zi, Rmn, Rmx of the PGON and PCON shapes.

[3] Names are modified comparing to the GEANT manual $1 \to x$, $2 \to y$, $3 \to z$ to clarify their meaning.

[4] The convention is to have the same first letter for any block within a whole module

This mechanism provides a simple and effective way to automatically generate the unique path to each GEANT volume, needed for the HIT package, without an additional user code.

2.5 Module header

The module header in DICE-95 is used to provide the Fortran declarations as well as the program maintenance information. It consists of the following *geant* declarations :

MODULE NAME	comment
AUTHOR	author list
CREATED	date or version
CONTENT	list of GEANT volume used
STRUCTURE NAME	{ list of variables }
+CDE,...	list of the KEEPs used.
Other Fortran declarations	

Note that:

- The first line should be the MODULE declaration, the order of other statements is irrelevant. The module name consists of a 4-letter detector code plus the module type code (GEO, DIG etc). It is also used to identify module input and output data structures (GEANT hits and digits, DETM - Master detector structure etc).
- The format of comment, author list and creation date is arbitrary, but their presence is mandatory.
- The CONTENT declaration should list all blocks used in the module.
- The STRUCTURE declaration groups together real variables or one-dimensional arrays, which are subject to potential change using datacards or should be accessible from external routines, for example at the reconstruction stage. Their usage is described in the next section.

Example:

```
      MODULE  GAASGEO  is the Geometry of the GaArsenid forward tracker
      Author     Rene Brun, Pavel Nevski
      Created    23 sept 94
      Content    GAAS, GDSi, GSij, GHij, GDij, GSUB, GASS, GELE, GSUP
      Structure  GAAG { Version, Ndisc, DrCounter, DfCounter, DzCounter,
                        TCKsubs, TCKsupp, DXele, DYele, Dzele  }
      Structure  GDSi { Disc, RIdisc, ROdisc, ZDisc }
      Real       Zdel,Rj,Zk,DR
      Integer    Idisc,j,k,N,ndv
+CDE,AGECOM,GCONST.
```

2.6 Data structure handling

A group of logically linked variables, which are declared in a STRUCTURE operator, is defined using the FILL statement:

FILL	NAME		! bank comment
	$variable_1 = value_1$		! explanation of $variable_1$
	...		
	$variable_n = value_n$		! explanation of $variable_n$

Note that:

- The structure name consists of 4 letters and is used as the ZEBRA bank name and as the prefix of its variables in the Fortran code.
- The order of assignments is irrelevant, but comments and explanations are mandatory.
- Other comment lines cannot interleave with FILL assignements.
- $value_i$ are Fortran expressions in case of a simple *variable* or a vector in the form $\{val_1, ...val_k\}$ for an array.
- When the FILL statement is executed by the *Atlsim* interface, the data are saved as a bank in the master detector (DETM) structure.

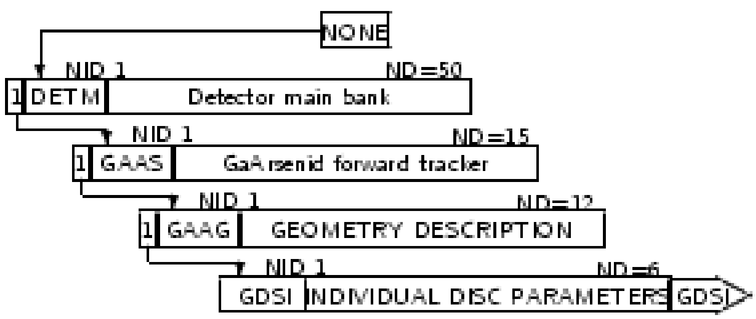
There may be two levels of data structures (banks) defined and used in a module: the structure name, defined by the first FILL operator, becomes the high level structure name. All structures with other names are considered as lower level structures associated to it.

Each of these structures may be a linear chain of similar banks, created by sequential FILL operators with the same name. They all are considered as instances of the same generic object, so at any moment only one selected copy of each structure is available. A typical usage of the high level structure is to provide different geometry versions of the same detector, the actual version been selected using the datacard input. Instances of the low level structures can be used to provide parameters for different components of the of the same detector.

Example:

```
 Fill  GAAG                       !  geometry description
          version    = 3            !  Annecy layout
          Ndisc      = 2            !  Nr. of discs (on each side)
          DrCounter = 5.3           !  DX (Dr) of a counter
          DfCounter = 2.6           !  DY (Dphi) of a counter
          DzCounter = .02           !  DZ (Thickness) of a counter
          TCKsubs    = .01          !  Thickness of the substrate
          TCKsupp    = 0.8          !  Thickness of the support
          DXele      = 3.           !  DX (Dr) of the electronics board
          DYele      = 2.           !  DY (Dphi) of the electronics board
          DZele      = .06          !  DZ (Thickness) of the electronics board
 Fill  GDSi                       ! individual disc parameters
          Disc = 1                  !  disc  number
          RIdisc     = 20.          !  inner Radius
          ROdisc     = 35.          !  outer Radius
          Zdisc      = 156.         !  Position along Z
 Fill  GDSi                       ! idem for next disc
          Disc = 2                  !  second disc
          RIdisc     = 25.          !  inner Radius
          ROdisc     = 40.          !  outer Radius
          ZDisc      = 185.5        !  Position along Z
```

Example: data structures produced by the previous example:



One can select the actual copy of the structure to be used by the program (an instance of the data structure) with the help of the USE statement :

$$\text{USE NAME } \textit{variable} = \textit{value}$$

Any variable from the corresponding structure can be used to select the current instance of the bank. The *value* may be any Fortran expression. Once the top level bank is selected with the USE operator, the descendent lower level banks are selected only within the same branch. Selected banks are re-linked at the first position of their top level banks, so that they always become default banks for any further selection. Also at that moment their content is changed by the standard datacard input.

Once selected with the USE operator, variables from the data structure can be referenced by the program in the form *BankName_Variable*. In this way they are easy to recognize among the other program variables (see first example).

This mechanism provides an easy and flexible way of the geometry versioning within each module.

3 CREATING GEANT HITS

In DICE-95 user does not need to write a detector specific routine to create GEANT hit structure and to fill it with a useful information. Instead, a *geant* statements with the HITS operator, called in a block describing a sensitive volume, is used to produce a relevant GEANT hit definitions and to steer their filling at the tracking time. This statement generates all necessary GEANT calls (see GSDET and GSDETH routines, HIT 100) with their parameters as follows:

- The *set name* is defined by the first 4 letters of the module name;
- The *detector name* is the name of the *geant* block ;
- Following the DICE standard, IDTYPE is taken as the detector number;
- The *name* argument of the HITS operator, (hit address) is the name of the volume used to identify the hit, usually the sensitive detector itself. The *Atlsim* interface finds the path to the selected volume using generic names of all higher level volumes and builds the NAMESV array. It also defines the number of branchings and the number of bits required at all levels (NBITSV array) to uniquely describe the path to each instance of the selected volume;
- For memory allocation defaults values of NHWI, NWDI = 1000 are used.

The definition part of the HITS operator contains a list of information quantities, measurements, which should be saved in each GEANT hit, and their packing in one of the form

$$measurement : N_{bit} : (min, max)$$
$$\text{or } measurement : bin : (min, max)$$

For a *measurement*, N_{bit} or *bin* are mandatory and limits are optional. At present the following variables are known as *measurements* to the *Atlsim* interface (the track point means here the middle point of the track segment producing the hit):

- x, y, z - local Descartes coordinates of the track point in the sensitive volume;
- $theta, phi, R(orRR)$ - local cylindrical (or polar) coordinates of this point;
- Cx, Cy, Cz - local direction cosines of the track segment;
- Cr - cosine of the angle between the track segment and the radius pointing to its center;
- TDR - closest approach of the track segment to the local z-axis;
- STEP - the length of track segment producing the hit;
- ELOS - the energy lost at this step;
- BIRK - equivalent energy of the calorimeter response (see PHYS 337);
- TOF - time of flight for this hit;
- ETOT - particle energy in the current point;
- LGAM - log_{10} of the particle Lorentz factor;
- ETA - pseudorapidity of the track point;
- USER - the hit quantity is calculated in a user function.

An integer number, following a *measurement* variable, is interpreted as N_{bit} - the number of bits for packing the variable values. 0 means that the value is a cumulative sum, occupying a full computer word. Due to the GEANT limitation 0 can be used only in last elements of the HITS statement.

If a *measurement* variable is followed by a real expression, it is interpreted as the packing bin size, and the number of bits, required for packing, will be calculated by the *Atlsim* interface.

If the user does not provide the limits explicitly, *min* and/or *max* are determined by the *Atlsim* interface using the volume dimensions.

Example:

HITS GASS X:12: Y:11: ELOSS:0:

In case of the USER element, a subroutine *XXXXSTEP(pointer,hit)* , where XXXX is the volume name, will be called to provide the *measurement*. This subroutine should be described as EXTERNAL in the module header. Its integer input argument *pointer* is the address of the hit description array (10 words, real) in the GEANT memory and it returns in *hit* the measurement. The format of this description can be found in the Appendix.

This option violates the data encapsulation principle as a user gets a direct access to the GEANT memory. It is not needed at present in DICE-95 and users are discouraged to use it until they are sure they really need it.

4 DIGITISATION

4.1 Detector response description

The detector digitisation , i.e. simulation of the response of individual elements of a given detector after tracking of a compete event, is done in a separate *geant* module.

A digitisation module has the header and the data handling part similar to a geometry module, but instead of blocks, describing detector geometry, it describes how a specific detector response in each separate element is produced, taken into account multiple hit overlap, noises, thresholds etc.

The content of the digitised information piece, the *digit*, should be described in the detector digitization module using the DIGI operator. This operator, similar to the HITS operator, creates necessary banks in the GEANT JSET structure.

DICE-95 creates digits in the *Atlsim* format, which is similar to the format of hits, but different from the one used for the standard GEANT digits. The difference is summarised below:

- The amount of memory used by the *Atlsim* digits in average is twice less then the one consumed by the standard GEANT digits.
- The transformation of the digitized measurements into integer numbers, representing packing bins, is done internally by the corresponding routines in the *Atlsim* library in the simular way as GEANT makes it for hits. When a user reads these digitised measurements back at the reconstruction stage, he gets them in the same coordinate system where they were "measured". This feature free the user from the necessity to transfer the packing constants to the reconstruction routines in a user code and eliminates one of the important source of the reconstruction errors.
- It is possible to introduce cumulative digitisations in the same way as the GEANT cumulative hits. If a non positive number was defined as the number of packing bits for a measurement, its value and the values of subsequent measurements will be summed, provided that the other digit parameters (track number,volume address,non-cumulative measurements) are the same.

4.2 Collecting all hits in a detector element

The *Atlsim* interface contains 4 integer functions (AgFHIT0, AgFHIT1, AgSDIG0, AgSDIG1) which provide the hit access and the digitisation storage service. Their execution and the print verbosity are controlled by the datacards in a way described later. If the operation was successful, the functions return the OK flag (0 value).

To select a hit set to be analyzed, a AgFHIT0(Cset,Cdet) function should be called, where Cset and Cdet are 4-letter names of a system and its sensitive detector. The function returns OK, if the selected set contains hits and the digitisation of this system has been requested by control cards, otherwise the digitisation should be abandoned.

If the address part of the DIGI set coincides with the address of the HIT set (same volume used as their address), this call also defines the output DIGI bank. Otherwise, if the *HITS* and *DIGI* detectors are different [5], the AgSDIG0(Cset,Cdet) function should be called.

After these initialization calls are successfully done, the *Atlsim* interface is ready to provide you sequentially with all hits in each detector element by performing the AGFHIT1(IH,ITRA,NUMBV,HITS) function. Here the output arguments are :

- *abs*(IH) will be on output the sequential hit number in the current detector element. A negative IH is used to signal the last hit in the detector element.
- NUMBV is an integer array, that will contain on output the list of volume copy numbers which identify the path to this detector element.
- HITS is a real array which will contains the measurements belonging to this hit.
- *abs*(ITRA) will be the track number having produced this hit. The negative ITRA is used to signal that other particles also contributed to this detector element.

The function itself returns OK until all hits in the selected set are used.

In this way in the digitization routine the user does not need neither to introduce arrays to accumulate the information from different detector elements in parallel, nor even to know the full number of the detector elements. Moreover, if a user needs to know the space position of a hit, he can simply use the GEANT routine GDTOM to translate a point in the current detector element to the Master Reference System, as the content of the necessary common blocks is restored by the *Atlsim* interface.

Finally, when all hits in one detector element are received, the AGSDIG1(ITRA,NUMBV,DIGI) function should be called to store the simulated digitisation. Here the input arguments are:

- ITRA is the number of the track that has produced this digit. A negative ITRA will be stored as zero.
- NUMBV is the address of volume to which this digitisation belongs.
- DIGI is a real array containing the digitised measurements.

Below you will find as an example a part of a calorimeter digitisation routine. It gets energy deposited in a set of tubes with arbitrary geometry and produce the digitisations as the energy sum in a standard η, ϕ presentation.

---

[5] This is the case, for example, in the tile calorimeter, where hits are registered and stored per tiles with certain (r,z) position, or in the integrated forward calorimeter, which contains tubes arranged in a certain (x,y) grid, but where the DIGITs should be stored per $\Delta\eta \times \Delta\phi$ bins.

Example: FWDC digitization loop:

```
     If (AgFHIT0('FWDC','FWAI') .ne. OK) Return
     If (AgSDIG0('FWDC','FWDC') .ne. OK) Return
     DO While (AgFHIT1(IH,ITRA,NUMBV,HITS) .eq. Ok)
        If (abs(IH) .eq. 1)  then  ! a new tube
           Esum=0
        endif
*       Accumulate energy just in one tube
        Esum = Esum+Hits(1)
        If (IH .le. 0) then        ! all hits in one tube received
*          translate NVL -> x,y,z -> eta,phi
           call GDTOM(zero,xyz,1)
           theta =  acos(xyz(3)/vmod(xyz,3))
           Eta   = -log(tan(theta/2))
           Phi   =  atan2(xyz(2),xyz(1))
           If (Phi .lt. 0) Phi = Phi + 2*pi
           DIGI(1)=Eta
           DIGI(2)=phi
           DIGI(3)=Esum
           If (AGSDIG1(ITRA,NUMBV,DIGI) .ne. OK)  Return
        Endif
     EndDO
```

A complete digitisation module of the tile calorimeter is shown in the appendix 3 as an example.

5 INTERACTIVE VERSION

The *Atlsim* library linked with an iteractive GEANT provides a unique possibility to study, modify and to debug the description of a new geometry. A special macro-command, make, compiles and executes dynamically in a stand-along mode any selected geometry module, existing in a separate file with the *.g* extension.

Using this program one can perform in particular the following operations with a single geometry module or with a complete ATLAS detector:

- CALL AGDROP('*') - to clear ZEBRA memory by droping all previously created banks.
- make *module-name* - to compile, link and execute interactively a module written in a separate file. The name of the file should be the same as the module name with the extension *.g* .
- DEBUG ON - to execute following modules in the debugging mode, with an increasing level of the *Atlsim* printouts. Most of the parameters of the created materials, media, rotation matrices and volumes will be printed.
- RZ/FILE 1 atlas.geom I - to read in a geometry file of the Atlas detector from the current directory.
- DRAW ... or DCUT ... - to draw different views of the selected system or its parts using GEANT graphics. In the debug mode (after DEBUG ON command) an isometric view of the system is drawn automatically.
- DTREE ... - to draw the logical tree of the GEANT volumes with their generated nicknames and dimensions.
- DISP detm detm.rz - to survey the tree of the created data structures, to navigate through them, to see the actual content of each created bank with its description, extracted from the module by the *Atlsim* interface.
- CALL AGDUMP('/DETM/*name**',0,'FH') - to produce HTML descriptions of all banks of a particular MODULE *name*.
- KINE Ikine Par(1-10) - to define parameters of simulated particles. In addition, last two parameters (9 and 10) limit the vertex position.
  Vertex position and spread can be defined by CALL AgSVERT(x,y,z,Sx,Sy,Sz) (default are LHC standard). Ikine =0 force simulation of a particle selected by Ptype. Ikine -1 corresponds to GENZ input and -2 corresponds to N-tuple input, -3 being reserved for a user input format. By default, corresponding file names ZEBRA.P, ncwn.hbook (hist # 4) and user.file. This can be overwritten by CALL AgNZOPEN(file), CALL AgNTOPEN(file,IDH) are CALL AgUSOPEN(file).

For more details the user is referred to the XINT section of the GEANT3 manual.

6 DATACARD CONTROL

6.1 Program control

Usual flags from the *MODE datacards are used by the *Atlsim* interface to control the geometry building (GEOM), hit saving in sensitive detector (SIMU), switching on/off of the magnetic field (MFLD), to allow detector digitisation (DIGI) or reconstruction (RECO). The control is done in a transparent way, so a user does not need to analyse this flags himself. The only interesting flag is GEOM, which is used also to select the detector version. This flag is available in a geometry module as %IGEOM variable.

The verbosity of the printout is also controlled by datacards. As the print requirements may be different not only from detector to detector, but also for different *stages* of the program execution, the actual print level is always produced as a product of the detector print flag, defined in the detector data card:

$$\text{*MODE 'XXXX' 'PRIN' } L_d \ \ldots$$

and of the current stage print flag, defined in the *stage* data card:

$$\text{*MODE 'YYYY' 'PRIN' } L_s \ \ldots$$

where XXXX are conventional detector system codes and stage codes YYYY can be 'GEOM', 'SIMU', 'DIGI', 'RECO' etc.

6.2 Print control

In general the action of the resulting print level $L = L_d * L_s$ is defined by the following strategy:

– 0 - no printout at all (same for L negative);
– 1 - minimal printout (not more than once per event);
– 2 - still reasonable amount of prints (up to 10 lines per event);
– 3 - you can tolerate it for a dozen events;
– 4 and more - debugging to find a problem.

Some particular cases for different stages are explained below.

6.2.1 GEOM - Geometry building stage

The print level decreases by one each time the program makes a jump into a next level block. So with small L you will get only general detector dimension, and with higher L you will get parameters of smaller detector pieces.

6.2.2 SIMU - Simulation stage

The printout, tracing particles, is done by the GEANT routine GDEBUG. This routines operates under the control of DEBUG and ISWIT data cards (see section BASE 400) and may produce a very abundant printout.

In addition, the *Atlsim* interface provides a possibility to tracing particles only in selected detector systems. A detector *MODE 'XXXX' 'DEBU' D data card is used to limit the maximal volume insertion level, where a call to GDEBUG is done. So with D=1 one will get the tracing only the system mother volume, and with higher D from its internal volumes. The total number of volume levels, where the tracing is done, is defined by the detector print level.

6.2.3 DIGI - digitisation stage

The detector *MODE 'XXXX' 'DIGI' d 'PRIN' L data card defines weither this detector will be digitized (d=1) or not (d=0). At the print level 3 and more, the total number of digitised hits will be printed for each event. If the print level is 4 and more, the output digi set will be dumped. If it is 5 and more, the input hit set is dumped.

6.3 Parameter input

The content of a data structure, defined in any module of DICE95, can be modified by a *DETP datacard. To modify a variable, user has to provide the name of the detector, the name and the value of the "use" selector of the desirable bank, and then names and new values of variables in the selected bank. All modification for the same detector should be done on the same *DETP datacard, which can be continued an several lines following the FFREAD rules.

Example. To modify the "Dx of the electronics board" in the example on page 5 on can use the following datacard:

*DETP 'GAAS' 'GAAG='3. Dxele='3.1

Note that dots are mandatory, but identificators are case unsensitive.

7 DOCUMENTATION AND DATABASE SUPPORT

As it has been already mentioned, when the FILL statement is executed by the *Atlsim* interface, the data are saved as a bank in the master detector (DETM) structure. At the same time the *Atlsim* interface creates the appropriate documentation banks for DZDOC package [2]. For each bank in the DETM structure the documentation banks contain the creation date, authorship information, the variable names and comments as well as the full information on the bank relationship.

All this information is maintained in a RZ-file detm.rz which can be analysed by the DZDOC package. Running its interactive version DZEDIT, users can get the full information on the created banks as well as to print a hardcopy of the current input data structure description. As the documentation RZ-file is updated automatically each time the program has been changed, this description is always up-todate.

It is possible to get with the USE operator not only versions of banks, defined directly in the module, but also to read them from the geometry data base, supported centrally.

Acknowledgments

Many people participated in discussions.. Authors want to thank Maxim Potekhin and Serguey Baranov, who were the first "test" users of the *Atlsim* package, Rob Veenhov and Sasha Vanyashin for many valuable comments on the draft of this paper. We are grateful to Rene Brun, who has initiated the present work and was extremely helpful to find a global view on its development.

References

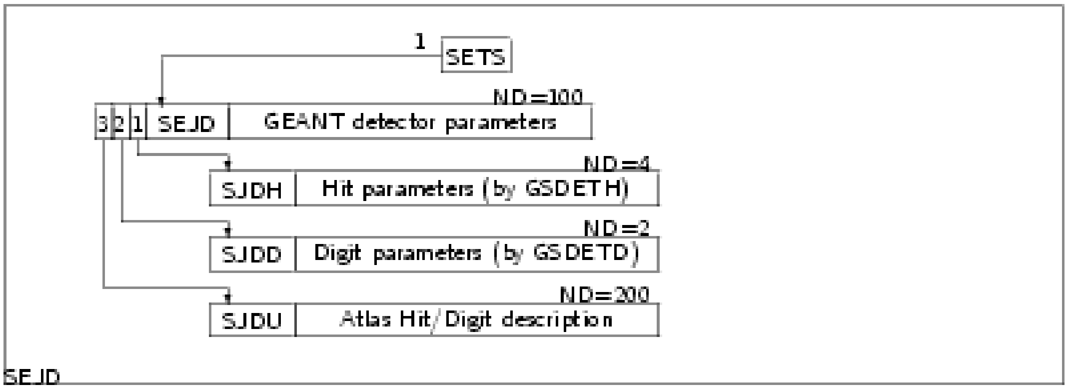
[1] GEANT - Detector Description and Simulation Tool. CERN Program W5013. Geneva, 1994.

[2] DZDOC - Bank documentation tools. In ZEBRA, CERN Program Q100/Q101. Geneva, 1993.

[3] HEPDB - Database Management Package. CERN Program Q180, Geneva, 1993.

Appendix 1: Description of the hit/digi banks used in GEANT3.

Hits and digitisations are stored in two symmetric sets. Internal names of sets, containing hits, end with H, while internal names of digitisation sets end with D. This is transparent to a user who always address them with the sub-system name. This allows to have different detector parameter banks (SEJD) for hits and digitisations.



SEJD GEANT detector parameters

```
---------------------------------------------- entered file at 13-Dec-94 14:59
Bank IDH  SEJD     GEANT detector parameters (filled by GSDET)
Author             R. Brun
Version            3.01
Store              /GCBANK/
Division           Constant
NL                   4
NS                   4
ND                 100
Up        SETS     -1
IO-Charac          10I / 1H 1I
             ---------- Description of the links  ----------
1         SJDH     - pointer to hit parameters
2         SJDD     - pointer to digitisation parameters
3         SJDU     - pointer to users parameters
             ---------- Description of the data words   ----------
1         Nwv      Number of words to store packed hit descriptors
2         Nv       Number of hit descriptors (Volumes + non-cumulative elements)
3         Nwh      Number of words per packed hit part
4         Nh       Number of cumulative elements per hit
5         Nwd      Number of words per packed digit part
6         Nd       Number of cumulative elements per digitisation
7         NWHI     primary size of hit bank
8         NWDI     primary size of digitisation bank
9         Npath    Number of paths through JVOLUM tree; (-1) after GsDETV call
10        Idm      For aliases only, IDET of mother detector
--REP level=1   Nv times
    11      NameVol  Name of a volume or a non-cumulative element
    12      NbitVol  Number of bits for packing its number
--REP level=1 -- End --
```

SJDH Hit parameters (by GSDETH)

```
----------------------------------------------- entered file at 13-Dec-94 14:59
Bank IDH  SJDH     Hit parameters (filled by GSDETH)
ND                 4 - per each hit element
Up        SEJD     -1
IO-Charac          / 1H 1I 2F
             ---------- Description of the data words   ----------
--REP level=1  Nh times :
    1       NameHit  Name of a cumulative element
    2       NbitHit  Number of bits for its packing
    3       origin   displacement for packing
    4       factor   scale for packing
```

SJDD Digit parameters (by GSDETD)

```
----------------------------------------------- entered file at 13-Dec-94 14:59
Bank IDH  SJDD     Digit parameters (by GSDETD)
ND                 2 - per each digi element
Up        SEJD     -2
IO-Charac          / 1H 1I
             ---------- Description of the data words   ----------
--REP level=1   Nd times :
    1       NameDig  Name of the digit descriptor
    2       NbitDig  Number of bits for its packing
```

SJDU Atlas Hit/Digit description

```
----------------------------------------------- entered file at 10-Jan-95 10:44
Bank IDH  SJDU     Atlas Hit/Digit description
Author             Pavel Nevski
ND                 200
Up        SEJD     -3
IO-Charac          -F
             ---------- Description of the data words   ----------
1         Iadr1    displacement for hit description part = 10
2         Nha      Number of hit descriptors (both in non- and cum. parts)
3         Iadr2    displacement for volume description part = 10+10*Nh
4         Nva      number of all volume descriptors (branching or not)
5         Iadr3    displacement for the free space = 10+10*Nh+3*Nv
6         Nvb      number of real volume branchings for NUMBV
7         option   1 - single step hit option (S in any hit element)
8         serial   sensitive volume serial number for this table
9         IdType   ATLAS detector number
10        Iprin    current print flag both for HITS and DIGI
--REP level=1  Nha times, j=10*ih
    j+1       hit       encoded hit name
    j+2       option    encoded hit option (R-rounding)
    j+3       Nb        number of bit requested
    j+4       Fmin      hit lower limit
    j+5       Fmax      hit upper limit
    j+6       Origin    Geant origin
    j+7       Factor    Geant factor
    j+8       Nbit      number of bit allocated
    j+9       Iext      address of the Geant user step routine
    j+10      Ifun      hit function code (1-18 at present)
--REP level=1  Nva times, k=10+10*Nha+3*iv
    k+1       Ivol      Volume of branching (pointer in JVOLUM)
    k+2       Ncopy     number of branchings
    k+3       Nb        number of bit needed
```

Appendix 2: example of a geometry module.

```
******************************************************************
MODULE    GAASGEO  is the  Geometry of the GaArsenid forward tracker
******************************************************************
Author     Rene Brun, Pavel Nevski
Created    23 sept 94
+CDE,AGECOM,GCONST.
   Content   GAAS, GDSi, GSij, GHij, GDij, GSUB, GASS, GELE, GSUP
   Structure GAAG { Version, Ndisc, DrCounter, DfCounter, DzCounter,
                    TCKsubs, TCKsupp, DXele, DYele, Dzele  }
   Structure GDSi { Disc, RIdisc, ROdisc, ZDisc }
   Real      Zdel,Rj,Zk,DR
   Integer   Idisc,j,k,N,ndv
* ----------------------------------------------------------------
 Fill  GAAG                        ! geometry description
          version   = 3            !  layout version (1-Cosiner,2-Panel,3-Annecy)
          Ndisc     = 2            !  Nr. of discs (on each side)
          DrCounter = 5.3          !  DX (Dr) of a counter
          DfCounter = 2.6          !  DY (Dphi) of a counter
          DzCounter = .02          !  DZ (Thickness) of a counter
          TCKsubs   = .01          !  Thickness of the substrate
          TCKsupp   = 0.8          !  Thickness of the support
          DXele     = 3.           !  DX (Dr) of the electronics board
          DYele     = 2.           !  DY (Dphi) of the electronics board
          DZele     = .06          !  DZ (Thickness) of the electronics board
 Fill  GDSi                        ! individual disc parameters
          Disc = 1                 !  disc  number
          RIdisc    = 20.          !  inner Radius
          ROdisc    = 35.          !  outer Radius
          Zdisc     = 156.         !  Position along Z
 Fill  GDSi                        ! same
          Disc = 2                 !  second disc
          RIdisc    = 25.          !  inner Radius
          ROdisc    = 40.          !  outer Radius
          ZDisc     = 185.5        !  Position along Z
*
    USE        GAAG   version=3
*   ---
    Create      GAAS
    call GSPOS('GAAS',1,'INNE',0.,0.,0., 0, 'MANY')
* ------------------------------------------------------------------
Block     GAAS   is GaArsenid forward tracker
    Material   Air
    Medium     Atlas
    Attribute  gaas          seen=0
    Shape      TUBE          Rmin=10     Rmax=50    dz=200
    Zdel = max(gaag_DZele,gaag_DZcounter+gaag_TCKsubs)  ! used later

    do idisc = 1,nint(gaag_Ndisc)
*
       USE        GDSi   Disc =idisc
*      ---
       Create     GDSi
       Position   GDSi   z=+gdsi_Zdisc              "forward"
       Position   GDSi   z=-gdsi_Zdisc  ThetaZ=180 "backward reflected"
    enddo
endblock
* ------------------------------------------------------------------
```

```
Block     GDSi   is one dics of GaArsenid

   Attribute  gdsi    seen=0
   Shape      TUBE    Rmin = gdsi_RIdisc                                      _
                      Rmax = sqrt((gdsi_ROdisc+gaag_DXele)**2+gaag_DYele**2) _
                      dz   = (gaag_TCKsupp+8*Zdel)/2

   Create and Position  GSUP  dz=gaag_TCKsupp/2

      DR = 0
      n  = nint( (gdsi_ROdisc-gdsi_RIdisc)/gaag_DRcounter )
      if (n>1) DR=(gdsi_ROdisc-gdsi_RIdisc-gaag_DRcounter)/(n-1)

      do j=1,n                       !  make radial divisions
         Rj = gdsi_RIdisc+(j-1)*DR
         Create      GSij
         do k=1,2
            zk=-gaag_TCKsupp/2-Zdel*(1+2*mod(j,2))+(k-1)*(gaag_TCKsupp+4*Zdel)
            Position  GSij    z=zk
         enddo
      enddo

endblock
* -----------------------------------------------------------------------
Block     GSij   is a sub-disc - one ring of overlapping counters
   Shape  TUBE   Rmin = Rj                                                   _
                 Rmax = sqrt((Rj+gaag_DRcounter+gaag_DXele)**2+gaag_DYele**2) _
                 dz   = Zdel

   Ndv = int (2*pi / atan(gaag_DFcounter/(Rj+gaag_DRcounter)/2)/4+1)
   Create      GHij
   position    GHij   z=-Zdel/2
   position    GHij   z=+Zdel/2    AlphaZ=360.0/(2*Ndv)
endblock
* -----------------------------------------------------------------------
Block     GHij   is a half of the ring - one plane of counters
   Shape       TUBE   DZ=Zdel/2
   Create      GDij
endblock
* -----------------------------------------------------------------------
Block     GDij   is a sector containing one counter
   Shape              DIVIsion   Iaxis=2  Ndiv=Ndv

   Create and Position  GSUB       x = Rj + gaag_DRcounter/2
   Create and Position  GELE       x = Rj + gaag_DRcounter + gaag_DXele/2
endblock
* -----------------------------------------------------------------------
Block     GSUB   is a GAAS substrate plus sensitive counter
   Component  H             A=1    Z=1   W=8
   Component  C             A=12   Z=6   W=5
   Component  O             A=16   Z=8   W=2
   mixture    Plexiglass    Dens=1.10
   Attribute  GSUB          SEEN=1   COLO=3
   Shape      BOX           dx=gaag_DRcounter/2   dy=gaag_DFcounter/2  _
                            dz=(gaag_DZcounter  + gaag_TCKsubs)/2

   Create and Position GASS  z=-gaag_TCKsubs/2
endblock
* -----------------------------------------------------------------------
```

```
Block     GASS   is a sensitive layer of the Ga Arsenid counter
   Component  GA           A=69.7  Z=31  W=1
   Component  AS           A=74.9  Z=33  W=1
   Mixture    GaArsenid    Dens=5.307
   Medium     sensitive    SteMax=gaag_DzCounter/5  Isvol=1
   Attribute  GASS         SEEN=1   COLO=4
   Shape      BOX          dz=gaag_DZcounter/2
+  - - - - - - - - - - - - - - - - - - - - - - - -
   HITS       GASS         x:11:  y:10:  El:0:
+  - - - - - - - - - - - - - - - - - - - - - - - -
endblock
+ ---------------------------------------------------------------------
Block     GELE   is an electronic board for one counter
   Material   silicon      A=28.09  Z=14  Dens=2.33  RadL=9.36  AbsL=45.5
   Attribute  GELE         SEEN=1   COLO=5
   Shape      BOX          dx=gaag_DXele/2  dy=gaag_DYele/2  dz=gaag_DZele/2
endblock
+ ---------------------------------------------------------------------
Block     GSUP   is a support for a whole GaAs disc
   Component  F            A=14.1   Z=7   W=.95
   Component  C            A=12.01  Z=6   W=.05
   Mixture    C_whiskers   Dens=.24
   Attribute  GSUP         SEEN=1  COLO=7
   Shape      TUBE         Rmin=0  Rmax=0  dz=0
endblock

end
```

Appendix 3: example of a digitisation module.

```
**********************************************************************
Module     TILEDIG   is the  DIGITIZATION routine OF THE TILE calorimeter
**********************************************************************
Author     Marzio Nessi
Created    10 Jan 94
Structure  Tdig      { Version,Scale,Emax,Etamax,Deta,Dphi}
+CDE,AGECOM,GCONST,GCUNIT.
*
   INTEGER     NV,NH,ND
   PARAMETER  (NV=10,NH=10,ND=10)
   INTEGER     NVL(NV), AgFHIT0,AgFHIT1,AgSDIG0,AgSDIG1, LTRA,IH,IR
   REAL        VMOD, HITS(NH),DIGI(ND), Esum,The,Eta,Phi,E,
               zero(3)/3*0./, xyz(3)
* -------------------------------------------------------------------
*
   If (FIRST) then
      Fill TDIG                    ! Digitization parameters
         Version= 1                !  version
         Scale  = 1.e6             !  ADC scale factor
         Emax   = 100              !  Max energy
         etamax = 3.0              !  rapidity limit
         deta   = 0.025            !  eta granularity
         dphi   = 2*pi/256         !  phi garnularity

      DIGI TBMA   eta:tdig_Deta:(-3,3),   phi:tdig_Dphi:(0,2*pi),
                  Eloss:0:(0,tdig_Emax)
      FIRST = .false.
  endif
* -------------------------------------------------------------------
    If (AgFHIT0('TILE','TBSA') # ok) Return
    If (AgSDIG0('TILE','TBMA') # ok) Return

    DO While (AgFHIT1(IH,LTRA,NVL,HITS) .eq. Ok)

       If (abs(IH) = 1)  then  ! a new tile
          Esum=0
       endif
*     Accumulate just energy in one tile
       E    = Hits(1)
       Esum = Esum+E
*                                         all hits in this tile received ?
       If (IH<=0) then
*                                         translate NVL -> x,y,z -> eta,phi
          call GDTOM(zero,xyz,1)
          the = acos(xyz(3)/vmod(xyz,3))
          Eta = -log(tan(the/2))
          Phi = atan2(xyz(2),xyz(1))
          If (Phi < 0.) Phi = 2*pi + Phi
          DIGI(1) = Eta
          DIGI(2) = phi
          DIGI(3) = Esum
          If (AGSDIG1(LTRA,NVL,DIGI) # ok)  Return
       Endif

    EndDO

END
```