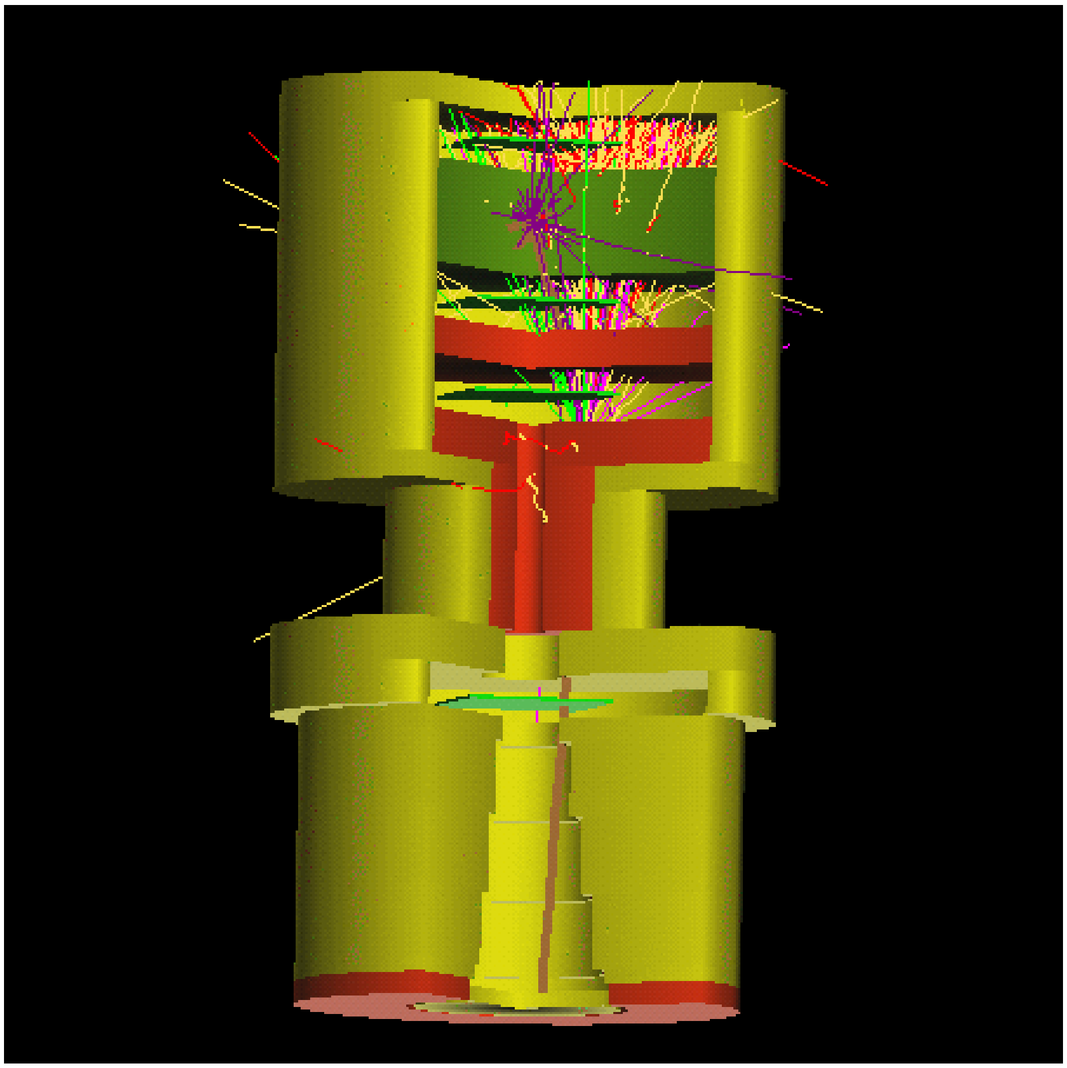
The Users Manual of Epics

K.Kasahara[*]

February 28, 2002



[*]Department of Electronic Information Systems, Shibaura Institute of Technology, Ohmiya, Saitama 330, Japan
e-mail: kasahara@icrr.u-tokyo.ac.jp

Contents

1 Introduction

Epics: Electron-Photon Induced Cascade Simulator in a detector. However, it can deal with hadrons and photo-production of hadrons. We need Epics, in spite of the availability of EGS4, GEANT and GISMO[1].

Although Epics has many good features, it cannot deal with the very low energy region precisely where we should consider the K-edge of the photoelectric effect, coherent scattering (Rayleigh scattering) and electron binding for the Compton scattering. Therefore, the energy region where you can play Epics safely for high Z materials is above about 100 keV which is, however, sufficiently low for many of applications[2]. Low energy particles die soon in a short distance, so even putting the minimum energy as low as a few 10 keV is expected to give reasonably good results. It is also to be noted that many applications utilize energy loss in a detector media, and it can be precisely measured by assuming the energy below the minimum energy ($\sim$ 100 keV) is absorbed in the media.

2 Important differences from the older versions

1. Geomview can be used to display detector configuration together with particle tracks.
2. In the detector configuration display, the 'sp' component is not displayed in default. If you want to avoid such default treatment, you should use 'hollow' in stead of 'sp'.
3. Some new volume-shapes are added: wavy plate, semi-wavy plate, light-guide-like form, chute (slide).
4. If you make a new application with a new config file, you must issue fordpmjet *config* to make dpmjet.inp and dpmjet.GLB files in the application folder (see p.38).
5. If you are going to use a config file newly which contains non pre-defined volume-shapes, you may go to *Epics*/Util and issue

   ./usenewvol *config*

   This will take care of all things needed for using the new volume-shapes. Therefore, you may forgot the old method which is, however, still available .
6. If you want to put a large nubmer of particles as incident, please see p.48.

3 Where is it?

You can find it by using the World-Wide Web. The URL is
`http://eweb.b6.kanagawa-u.ac.jp/~kasahara/`
After you are connected to the above site, you will easily find the Epics home.

4 Installation

4.1 Platforms

To be able to use Epics, the **Cosmos library (uv6.30 or later)** is indispensable. That is, the platforms where Epics can run are the same as for Cosmos. The Cosmos library is mainly used to treat hadronic interactions.

[1]EGS cannot deal with hadrons. They cannot cope with the 100 TeV energy region. The Landau-Pomeranchuk-Migdal effect is not properly treated in GEANT (GEANT4 seems to be improved.) GISMO uses Gheisha as a hadronic interaction model (GEANT's standard seems to be also the same) which is questionable.

[2]For low Z materials, this will be relaxed much. In some case such as synchrotron photon emission in vacuum, the minimum energy can be much lower.

One caution for NEXTSTEP is that it cannot use fritiof1.6 and dpmjet3 as a hadronic code; Cosmos on NEXTSTEP can run with dpmjet3 embedded in the library if it dose not use dpmjet3. However, Epics on NEXTSTEP cannot run if dpmjet3 is included in the Cosmos library even if it bypasses to use dpmjet3. You may exclude compliling dpmjet3 by giving a flag in Cosmos/cosmos/Zcondc.h. However, NEXTSTEP can be used to check detector configuration and displaying it together with particle tracks even dpmjet3 is included in the Cosmos library.

4.2 Installation

The installation of Epics is simple and very similar to that of Cosmos. Since Epics uses the Cosmos library, the first task is to build it. Then, next is to create the subroutine library of Epics. To do so,

1. Get the Epics source code.
2. Fix a directory where you want to put Epics. We shall express this top directory as "*Epics*".
3. Unpack Epics source in *Epics* (Say, by "`zcat Epics.uv6.00.tar.gz | tar xvfp -`")
4. In *Epics*, find site.configXXX appropriate for your environment and copy it to site.config. For instance, if you use a Intel Fortran Compiler on Intel PC Linux,

   ```
   cp site.configPCLinuxIFC site.config
   ```

   This is the same as Cosmos. For Digital (Compaq) Unix Alpha, use site.configDECALPHA2.
5. Issue

   source Scrpt/sevi

   to set the environment for Epics. Unlike the Cosmos case, **don't** put the Scrpt directory in your command search path, if you continue to use Cosmos as a tool for simulating cosmic rays in the atmosphere.
6. Issue

   make clean ; make

This will create *Epics*/lib/PCLinuxIFC/libepics.a, if you have specified to use the Intel Fortran Compiler for Intel PC Linux.

5 Touching Epics

It is time to confirm your installation of Epics by a test run; *Epics*/`UserHook/FirstKiss` is the directory containing the stuff for it. The file 'config' there expresses a detector consisting of 4 box type components surrounded by another box called the 'world'. Each inner component is made of box-shape scintillator with a thin lead plate attached to it; the boxes are inclined with different angles (see Fig.1).

- Be sure you are in *Epics* and already issued the command "source Scrpt/sevi". This is needed **whenever you open a new window to use Epics**.
- First, let's see the detector configuration which is used in this test run.
  - Go to *Epics*/Util.
  - `make drawConfig`

    or

    `make -f drawConfig.mk`

    This will create an executable, "drawconfig$ARCH". Here $ARCH stantds for ARCH defined in your site.config; therefore, for example, you will get drawconfigDECALPHA if you are using digital unix (True64) with DEC Fortran.

– Issue[3]
`drawconfig`
Then, the system will ask you to input the path to the detector configuration file. You may input
`../UserHook/FirstKiss/config`
(Since this is default, you may simply push the return key). Then you will see the following lines on the display.

```
 SELECT MENU NUMBER
 1) Make data for display (=1)
 2) Specify transparent surface/ranges
 3) Specify components by number
 4) Specify components by media
 5) Specify components by mother/daughter relation
 6) Specify components by structure
 7) Reread current config file
 8) Show config info
 9) Resume default setting
10) Read new config data file
11) Media<->color mapping
12) Save/recall display condition/change default
13) Reverse Z-coord
14) Level control
 0) Exit
```

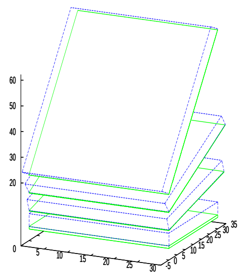


Figure 1: First default plot

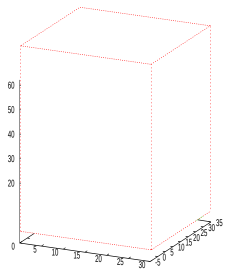


Figure 2: Adding the world

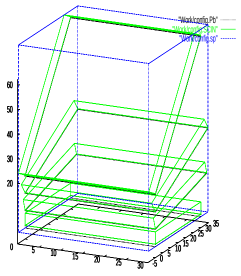


Figure 3: No hidden mode

– Here, you normally select a certain menu number to fix how you would like to display the detector. You may repeat such a process as many times as you want, and finally you have to select menu 1 which actually create data for display.

– Select menu number 1. (So you have created data for display with the default display condition. The data has been stored in `./Work` as gnu, config.Pb etc.)

– Now you have two choices to display the detector configuration: One is to use gnuplot, i.e., same as in the older versions, and the other to use Geomview, of which use is strongly encouraged. However, currently data for Geomview display is created using data for gnuplot so that this drawconfig process must be learned even you are going to use only Geomview.

– Regardless of whether you are going to use gnuplot or Geomview, open a new window. Then, for gnuplot, go to *Epics*/`Util`. For Geomview go to *Epics*/`Util/Geomview`. The Geomview interface is given in a separate paper. Here we show the gnuplot interface.

[3]you can invoke drawconfig$ARCH directly, but it is recommended to use the command 'drawconfig' which invokes drawconfig$ARCH internally; 'drawconfig' does some other jobs except for invoking drawconfig$ARCH, and this is essentailly important when you use GEOMVIEW.

- – Invoke gnuplot at *Epics*/`Util`.
  Issue[4]
  `call "config"`
- – The above command will show a picture as in Fig.1
- – This view lacks the world component made of 'sp' since the default display dose not show the 'sp' component (which is almost equivalent to the vacuum). To verify it's existence, let's draw all the compoent.
- – Move to the 'menu window'component (where you issued the 'drawconfig' command). The window should show
  Enter return (or take thin menu)
  where you can simply push the return key to display the menu again or, if you remember the menu, you can directly enter a menu number here.
  Select menu 4 and then use '**2** Select some media to be hidden'; hit return since you don't want to hide any media, and then, select menu 1.
  Upon going back to the gnuplot window, you may redraw the result by call 'config' to see Fig.2
- – To see the inside of the world box, we may use
  call "xconfig"
  set key
  rep
  which brings us Fig.3. 'set key' is to see the color and media correspondence.
- – It may be worthwhile knowing and doing the following.
  - ∗ Rotation must be specified by the 'set view' command in gnuplot.
  - ∗ The commands issued in gnuplot are not related to Geomview at all. Only the specifications in drawconfig affect Geomview display.
  - ∗ You can *make transparent* some part of the objects such as box, sphere, cylinder etc. This enables us to look inside the detector.
    This is managed by menu 2 where you can specify the transparent azimuthal and polar angle region of a spherical object. For a cylindrical object, you can also specify the transparent azimuthal angle region, and the top and bottom part of the cylindrical object be transparent or not. An example in which some azimuthal angle region is transparent for cylindrical objects is given in the front cover of the manual.
    For a box-like object, you can specify each face be transparent or not. For example, after selecting menu 2, you can skip angle specification by simply putting '/'. For transparent face specification of the box, try '011100' which means top, bottom and front face be transparent (you will be able to know which digit corresponds to which face after selecting menu 2).

- Next, let's run Epics. In *Epics*/`UserHook/FirstKiss`, issue
  `make clean; make`
  If everything is good, an executable, "**sepics$ARCH**", will be created[5]. Here $ARCH has the same meaning as explained earlier for drawconfig; in the manual we will be omitting this part and simply use sepics hereafter.
- If you have no directory /tmp/foo (where 'foo' stands for your login name), make it[6] This is the default directory in which Epics generates trace information of particle tracks. If you generate several events, trace1, trace2, ... will be the file names.

[4]Internally `load "Work/gnu"` is used. In some system (e.g., HP), the '`call "config"`' command sometimes fails to draw all the components. In that case, use the same command again.
[5]Other options for make: make veryclean and make rmtemp are available for use.
[6]The directory can be specified by TraceDir in 'epicsfile' .

In normal Epics runs which do not take trace information, you don't need this.

- Issue[7],

  `sepics < FisrtInput`

  Then, in a few moment, you will see some output on the screen; some of them are the energy loss at some detector components. **Congratulations!** (May they all be not error messages.)

- As noted on the screen, you can plot the tracks of particles together with the detector configuration by gnuplot or by Geomview. For the latter see the separate manual.

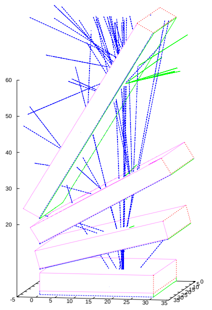


Figure 4: Full hidden mode with all particles

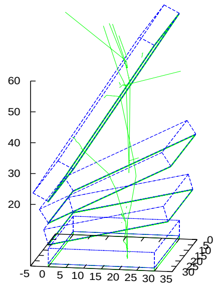


Figure 5: Transparent mode with charged particles only

  - It is recommended to use two windows; one for gnuplot and the other for 'drawconfig' by which you can change the display condition quickly.
  - Go to *Epics*/Util, and draw the detector configuration. That is, in the gnuplot mode, issue

    call "config"

    If the detector has very large number of components, the hidden mode is not recommended so that you may use,

    call "xconfig"

    alternatively.
  - To show all the cascade tracks together with the detector with the 'hidden mode', use[8],

    call "allh"

    This will create a figure like in Fig.4[9].

  - To cancel the hidden mode and display only charged particles, you may issue[10] (Fig.5),

    call "xconfig"

    call "only+−"
  - The 'call'able command files are listed in Table 1. An event number may be optionally appended.

[7]Remember actually you have to type sepcisDECALPHA etc

[8]`set hidden`, `rep "<awk -f hmode /tmp/$USER/trace1" w l` , `set nohidden` are used internally. Five events have been generated in this example, you may try call "allh" 5 to show the 5-th event

[9]Due to the limitation of gnuplot, the hidden mode cascade tracks may not appear in one and the same color. If a cascade has large segments, it is normally not recommended to use the hidden mode since display needs very long time

[10]Internally `rep "<awk -f cmode /tmp/$USER/trace1" w l` is used for `call "only+-"`

– If you have trace data for a number of events, a shell script "slide" will be useful to show such events consecutively. It can be invoked from the command line (at Util) as
./slide
then you will know the usage. Try
./slide 1 5 90 90 3
to show the first 5 events with 'set view 90,90' and a 3 second pause. The 'slide' command can be invoked from within gnuplot as
! slide 2 4

only+−	shows only charged particles with no hidden mode
only+−h	shows only charged particles with hidden mode
only0h	shows only neutral particles with hidden mode
only0	shows only neutral particles with no hidden mode
sort	shows only charged particles. Different colors for different particle codes
sort+−	similar as above, but +− charges are further distinguished.
allh	show all particles with hidden mode
all	show all particles with no hidden mode
oslide	show sharged particles of 1∼20 events with nohidden mode

Table 1: 'call'able command file list for track display. Optional event number may be added.

- 'call "slide"' is now obsolete, which existed before uv6.30. However, this can be used as 'call "oslid"'. Before calling this, you must give 'a =−1' or 'a = some positive integer'. This is to show a maximum of 20 events consecutively. If a=−1, system stops after displaying each event and waits for your hitting 'return'. If a=n, system stops n sec after displaying each event.

- The trace data format is

```
x y z  code   E  charge
```

"code" is the particle code (see 18.5), E the energy in GeV, "charge" the particle charge. Two blank lines indicate the end of a continuous track line. Trace file name will be 'trace1', 'trace2', ... for event 1, 2,... With this knowledge you may draw only, say, charged pions.

- The "config" file in FirstKiss contains a line like

```
#eq   cos15  9.659258263E-01
```

This equates a symbol 'cos15' to 9.659258263E-01 (which is cos(15/degrees) of course). After defining a symbol, you can use it in the main body freely. The present version **dose not allow to use a mathematical formula** in the right hand side (only a numeric value is permitted). Further details of the config file will be discussed later.

6 How to make your own application

6.1 Overview

To use Epics, you must have at least two files: One is for controlling physical processes involved in cascade development such as the minimum energy of particles to be treated (usually the file is denoted by 'epicsfile'), and the other the detector configuration (media and geometrical structure) where cascade is to be developed (usually denoted by 'config').

Epics itself has no main routine, so that you may write a main program, input particles to Epics and pass the control to Epics. Then, Epics develops cascade in the given environment and calls some appropriate user interface routines.

However, it is a troublesome task for the user to write such procedures for each application. Therefore, standard procedures are available for use. This is called Sepics (= Standard Epics) of which code is in the *Epics*/Sepics directory. Then, you may simply write user interface routines of which the template is given in the *Epics*/UserHook/Template directory. This method is similar to Cosmos. Sepics reads one file (example is 'FirstInput' in FirstKiss), in which you specify 3 to 4 files: One is for Sepics in which you have to give data such as the number of events to be simulated, the file name in which you give primary spectrum data (this file is usually denoted by 'sepicsfile'). The other two mandatory files are for Epics as explained above ('epicsfile' and 'config'). (We shall simply use the term, Epics, if distinction between Epics and Sepics is not important.). The last optional file is denoted by 'param' which is mainly used to control hadronic event generators. It's organization is the same as for Cosmos, but only few parameters are recognized in Epics.

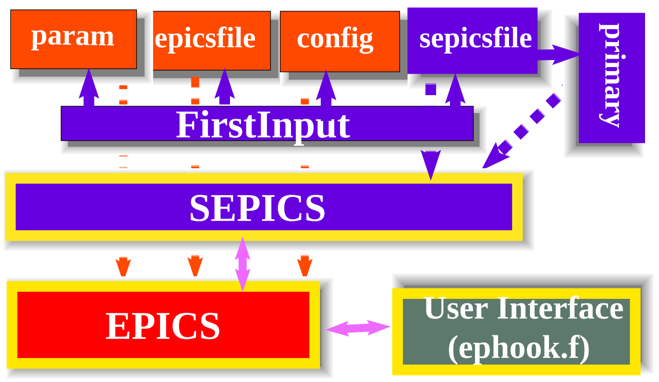


Figure 6: Codes and files in Standard Epics. We need two more files for dpmjets.

In summary, in normal Epics applications, you will need 5 to 8 files:

1. **FirstInput**: To show the names of epicsfile, config, sepcisfile.
2. **epicsfile**: Specifies basic parameters for an Epics execution. In most of cases, you need not touch this. In some case you may set/reset **Trace** parameter.
3. **config**: detector configuration file.
4. **sepicsfile**: To specify your job conditions not listed in epicsfile; typical ones are the number of events to be simulated, input particle position, angle etc. The file name containing primary particle information is also listed in this file.
5. **primary**: The primary particle information is given in this file. The format is the completely the same as for Cosmos. You may specify a monoenergetic particle or a number of different species of particles with energy spectra.
6. **param**: The same as the parameter file used in Cosmos. This is used to specify the hadronic interaction models. If the default model is to be used, you need not give it in 'FirstInput'.
7. **dpmjet.inp and dpmjet.GLB**: Used for dpmjet3. These two can be created by issuing fordpmjet configfilepath where configfilepath is the path to the config file.

If Sepics is not suited for your purpose, you can write your own main program, and call Epics routines yourself. Although very rare, this type of application may be needed if you have a large program already and you want to manage cascade showers in that program.

For most of applications, Sepics is enough and you have to/may do the following:

- In *Epics*/UserHook, you may make a copy of the Template directory and give the copy an appropriate name suited for your particular application. In that new directory,

- Modify ephook.f as you need.
- Edit 'epicsfile', 'config', 'sepicsfile' and 'param'. Give primary data in 'primary'.
- 'make clean' and 'make' to compile ephook.f and make an executable, 'sepics'.
- sepics < FirstInput

  Normally you may redirect your output like

  ```
   sepics < FirstInput > /tmp/1TeV.d
  ```

- The default interaction code, dpmjet3, includes the target break up process while others do not. However, it needs some time for initialization so that you may specify

  IntModel='"nucrin 4.5 "fritiof1.6" 9 "fritiof7.02"'

  when you are testing your program. The model shown above is also good at least at < 100 GeV.

6.2 Brief Summary of Input

In any Sepics applications, you must prepare the following files:

- One containing the primary spectrum data ('primary'). **The format and all details are explained in *Epics*/Data/Primary/sample.d**.

  The format is completely the same as Cosmos. This means you can use Cosmos utilities to show spectrum shape in order to confirm the input data, or sample the spectrum actually and see the result.

- A data file to control physical processes of Epics ('epicsfile'). This template is in *Epics*/UserHook/Template/epicsfile.

  Among parameters defined there, you may sometimes need to change `EminElec, EminGamma, KEmin` and `Trace`. You rarely need to change others. `Trace` may be set to "t" only if you want to have a look at particle tracks; this mode produces a huge amount of data. Disk file for the trace data is automatically opened if Trace is "t".

  Some are obsolete but kept for compatibility with the older versions.

- A data file to specify detector configuration ('config').

  Epics/UserHook/Template/config

  is a simple example. The details will be described later.

  The files of the older versions (<uv5.00) are usable without any modification, although there are a lot of extensions in the current version.

- A data file to specify the Sepics conditions ('sepicsfile'). This template is

  Epics/UserHook/Template/sepicsfile

  Some (job time control etc) are currently not used. You have to give a name of a data file in which primary spectrum data is given to *PrimaryFile*. Also, frequently used ones are *Nevent* to specify the number of events to be generated, *CosNormal* to specify the incident angle region, etc.

- A data file to control interaction model selection('param'; optional). This template is *Epics*/UserHook/Template/param. The format is completely the same as for Cosmos but you may think only IntModel is recognized in Epics. For example, 'Trace' to specify how to take trace data or 'PrimaryFile' to specify the primary are neglected.

- A small file ('FirstInput') to show the above 3 to 4 files and the job continuation condition (The "job continuation condition" is currently not used). A template is

  Epics/UserHook/Template/FirstInput.

Although we will use 'FirstInput', 'primary'... etc for the file names explained above, the names can be arbitrary given by the user.

6.3 Config file

The old 'config' files (<uv5.00) can be used as they are; the impatient user may skip this section if he/she uses only the old 'config' files forever.

The unit of the length in the config file is cm.

6.3.1 A simple pile configuration

Most of detectors used in cosmic ray physics use a pile of box-like detector components with the same width and depth, but with various thickness. The following list is such a simple example.

```
#  any comment may be placed before ------ line.
#  type    matter   c   de   maxpath ch#   / x  y   z   a   b  c
---------------------------------------------------------------------
#eq     a    40
#eq     b    50
1  box     Pb      0    0            / 0  0    0    a  b 1
2  box     SCIN  0    2     0  1  / =  =   +   =  = 4
c        This is big space
3  box      sp      0    0          / =  =   +   =  =  10
!          you can use ! also for comment line
#   #  is also for a comment line (but there are exceptions)
4  box     Pb      0    0          / =   =   +   =  =  0.5
5  box     SCIN  0    2    0   2  / =  =   +   =  =  4
/*
6  box     Pb       0   0            /  =  =   +    =  =    .5
7  box     SCIN   0   1     0  3 /  =   =  +    =  =  10
*/
---------------------------------------------------------------------
```

The main body of the configuration must be contained in two '------ ' lines. Any lines before or after these two lines are ignored.

The #eq assigns a numerical value to a symbol; 40 to 'a' and 50 to 'b'. They can be used instead of the number.

Each consecutively numbered line (from 1) describes a detector component. In the above example, only the box shape is defined as components ('type' column). The entry in the 'matter' column specifies a medium of the component (Pb for lead and SCIN for plastic scintillator. For available media, see 18.1). In the next 'c' column must be specified a number (0, 1, 2, or 3); 1 specifies that the user wants to count particles when they enter the component[11], 2 to count particles exiting from the component, 3 to count particles both entering and exiting. The 0 means no count is intended. If non 0 entry is given, the user interface routine, 'userbd' (in ephook.f), is called from the system at appropriate timing and the user may count particles there, or do some other business such as the generation of **transition radiation.**

The next 'de' column is to count energy loss in the component. If it is ≤ 0, no energy loss count is intended for that component (internally Epics of course takes care of energy loss) , while 1 or 2 must be given if the user wants to count energy loss in that component. Usually, a charged particle moves in a component with a zigzag track (consisting of a number of segments). For positive entry, the energy loss count routine, 'userde' (in ephook.f), is called whenever the charged particle moves along each of such segments. The energy loss informed to the user is as follows: If 1 is given, the average energy loss is given. If 2 is given, the Landau-Vavilov-Gaussian fluctuation of energy loss

[11]The incident particle at the initial position in any component is regarded that it is already there and not regarded as entering to that component.

a la Urban is considered. The average energy loss in the latter case is the same as the former case. If the energy loss information is needed, it is recommended to give 2. For the details, see p.47.

Normally, the 'maxpath' column may be kept blank or you may give 0; in that case, Epics takes some appropriate value automatically. If you want to limit the maximum path length of particles traveling in the component by some reason or other, you may give it in the 'cm' unit. (If you use very thin foil such as 10 μm or less, you may try various 'maxpath' and examine the results. The maxpath value may be order of $\sim 1/5$ of the layer thickness).

In the next field you can give an optional channel number of the component(see p.48).

The next '/' is mandatory.

The 'x y z' columns specify the origin of the component in the 'world' coordinate. The world coordinate is the Cartesian coordinate where each detector component is placed. Each volume-shape ('box' etc) has its origin in its own local coordinate system. Such a local origin is mapped to this origin (parallel translation).

The 'a b c' columns are for giving values describing the volume attribute. In the case of 'box', there are 3 attributes, i.e., the lengths of 3 edges (See Table 2 and Fig.8).

A (nestable) pair of `/*    */` can be used to indicate that lines inbetween are comment.

6.3.2 pile-up and pile-down

Except for the first component, you can use '=' , '+' or '~' besides a number or #eq-uated symbol. The most frequent use of '+' would be in the z-column when you pile up components, that is, the value for '+' becomes (the z-origin of the previous component + its 'z-thickness'); the '+' in the z-column of the 3rd component of the above example is replaced by 5. The '=' sign means that the value there should be the same as the corresponding value of the previous component.

```
#  pile up
------------------------------------
1 box  Al  0 0 0 / 0 0 0   3 3 1
2 box  Fe  0 0 0 / 0 0 +1  = = =
3 box  Fe  0 0 0 / 0 0 +1  = = 1.5
4 box  Fe  0 0 0 / 0 0 +1  = = 2
5 box_w sp 0 0 0 / 0 0 0
-------------------------------
```

```
#  pile down
---------------------------------------
1 box  Al  0 0 0 / 0 0 -1  3 3 1
2 box  Fe  0 0 0 / 0 0 ~-1  = = =
3 box  Fe  0 0 0 / 0 0 ~-1  = = 1.5
4 box  Fe  0 0 0 / 0 0 ~-1  = = 2
5 box_w sp 0 0 0 / 0 0 0
--------------------------------
```

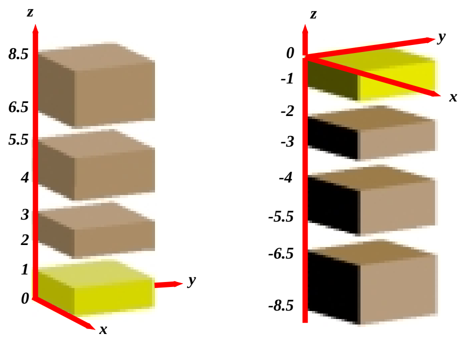


Figure 7: Left: pile-up configuration and Right: pile-down configuration corresponding to the configurations shown above.

The '~' sign implies the negative sign '−'. By using this convention, you can pile down the components. Symbols '=', '+' or '~' may be followed by a numerical value or a symbol (defined by #eq) (like '=10', '~r', '+xyz' etc), then the numerical value (or the number equivalent to the symbol) is regarded as an offset to be added to the value which would result if they were absent. The two examples shown above are piled-up and -down boxes with 'sp' components inbetween. We note that we didn't give 'sp' boxes except for the world. Also we note that if we put each 'sp' component inserted at each layer explicitly, we didn't need the world in this particular example. For more complex detector configuration, we will come back shortly.

6.4 sepicsfile

By this file, you determine the simulation conditions. The details are given in *Epics*/FirstKiss/sepicsfile. Some examples are given for explanation.

6.4.1 Fixed position and angle of the incident particle

We assume that the detector configuration is a 'pile-up' type.

CosNormal (0.9, 0.9) We want to fix the cosine of the zenith angle of the incident particle to be 0.9, but don't care about the azimuthal angle.

InputA 'is' The input angle is isotropic (default).

InputP 'fix' The incident position is fixed (default).

Xinp 15; Yinp 15; Zinp 0 The incident position is at (15, 15, 0)

6.4.2 Fixed position as a special case of the uniform distribution

The same input condition can be realized in the following way, which has wider applicability.

CosNormal (0.9, 0.9)

InputA 'is'

InputP 'u+z' This actually specifies that the incident position is uniformly distributed within a box region specified by Xrange, Yrange and Zrange parameters. The incident is directed to the positive z with the zenith angle given by CosNormal.

Xrange (15,15); Yrange (15, 15); Zrange (0,0) We must specify the incident position to be the fixed point.

If we use 'pile-down' structure, we would give, **InputP 'u-z'**.

6.4.3 Uniform on a sphere

To input incident particles around the detector isotropically, we may use

InputP 'usph' Input position is uniform on some region of the world sphere.

Xinp 90 In this case, this specifies the polar angle, $90°$.

Yinp 0 This specifies the azimuthal angle. But in this example, it can be arbitrary because Xinp is $90°$.

Zinp 90 (Xinp, Yinp) specifies the 'north' pole. The sampling region is within $90°$ around the north pole, i.e., on the hemi-sphere.

We note that, if InputA is 'is', the zenith angle distribution is $\cos\theta d\cos\theta$ at a given sampled point. You can specify $\cos^n\theta d\cos\theta$ distribution by giving 'cos n' to InputA.

6.5 Config File (continued)

6.5.1 Pre-defined volume-shapes

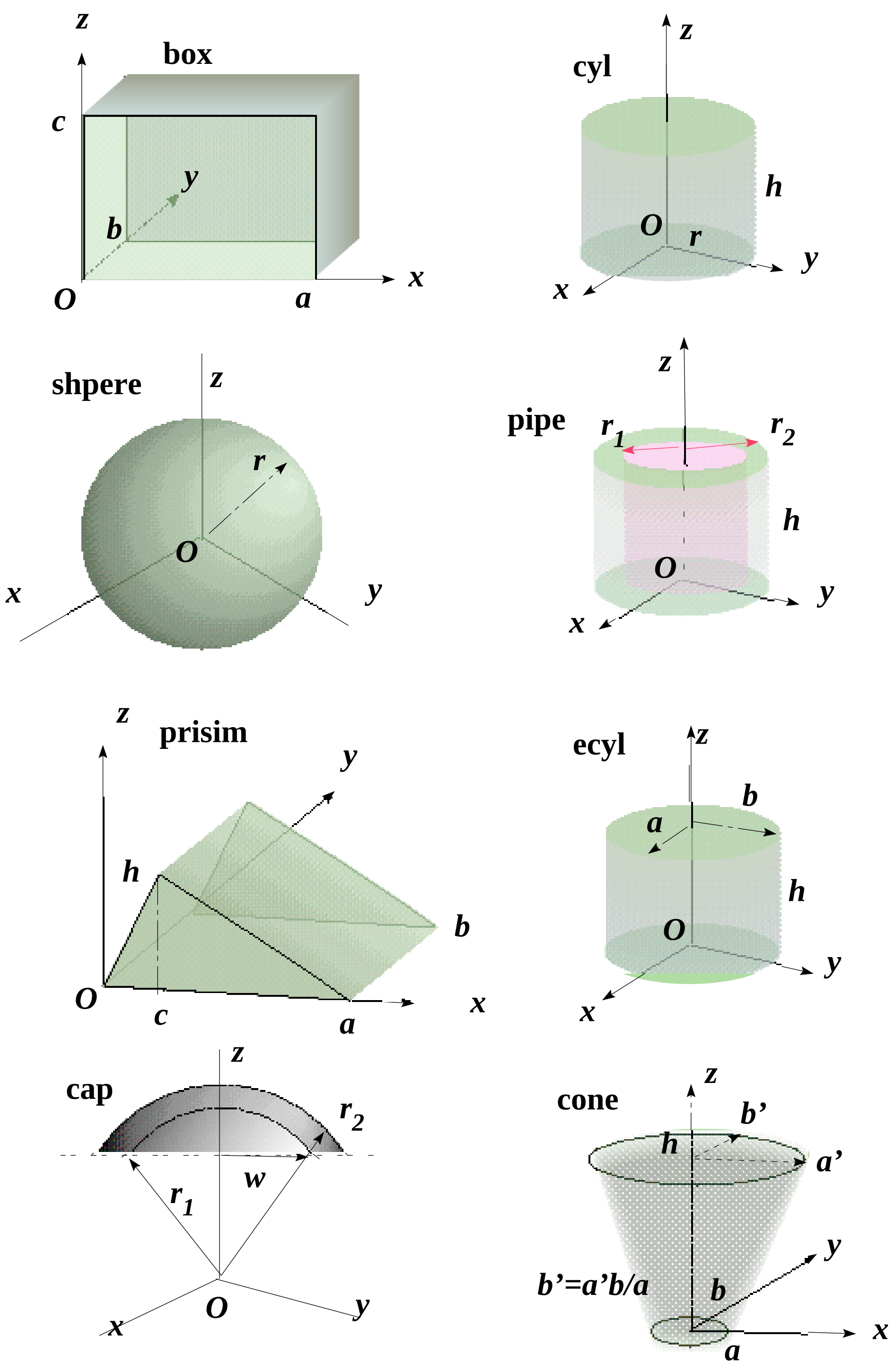


Figure 8: Canonical form of the predefined 8 volume-shapes

volume-shape	attributes	optional direction cosines	notes
box	a b c	ax ay az bx by bz	direction cosines are for a and b.
cyl	r h	zx zy zz	direction cosine is for z axis
sphere	r		
pipe	r1 r2 h	zx zy zz	direction cosine is for z axis
prism	a b c h	ax ay az bx by bz	a b c h can be negative. direction cosines are for a and b
ecyl*	a b h	zx zy zz	elliptic cylinder. direction cosine is for z axis
cap*	r1 r2 w	zx zy zz	if w $<$ 0, |w| is used but r1 and r2 are directed to z $<$ 0.
cone*	a b h a'	zx zy zz	direction cosine is for z axis

Table 2: Volume attributes and optional direction cosines for inclined form. The last 3 volumes (with * attached) are not a predefined volume-shape but an example of user-defined new volume-shape. There are lots of other new volume-shapes in the *Epics*/prog/NewVol directory. They are explained in a separate paper.

Figure 8 shows the canonical form of predefined volume-shapes. The number of volume attributes changes from 1 for sphere to 4 for cone and prism. These must be listed after the 'x y z' column in the config file. If the volume is inclined, the direction cosines showing the inclination must follow the volume attributes.

The sum of 3 direction cosines must be 1.0 within ∼5 digit accuracy otherwise an error message comes out and the system stops when reading the config file. The direction cosines are re-normalized in the system so that the sum be exactly 1.0 within the double precision accuracy.

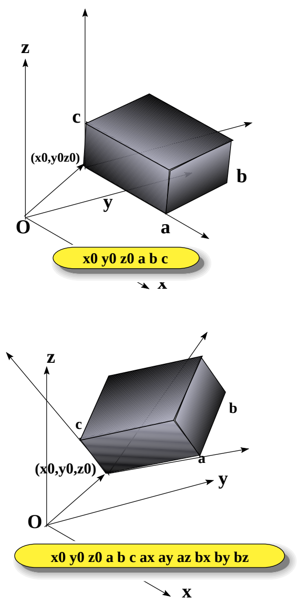


Figure 9: Parallel translation(upper) and rotation followed by a translation (lower) of a box

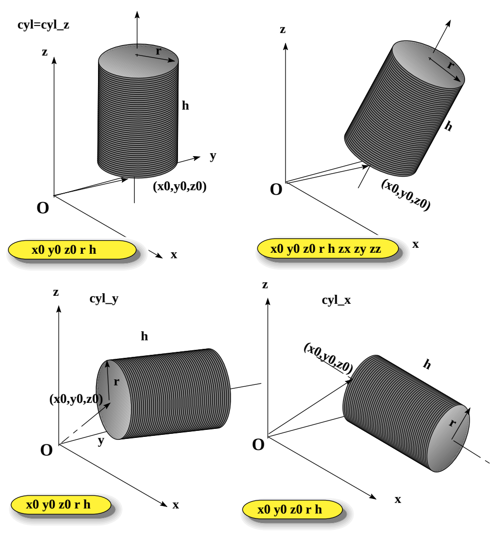


Figure 10: Translated and general cylinders (upper) and cyl_y (lower left) and cyl_x (lower right)

6.6 Non canonical form

For a box, the general non canonical form is shown in Fig.9. The yellow 'round table' shows the origin, attributes and optionally required direction cosines. Examples of inclined boxes can be seen in the config file of FirstKiss.

The cylinder case is shown in Fig.10. Since it is bothersome to give a 90-degree rotated volume by specifying direction cosines to the canonical form, you can use cyl_x, cyl_y, cyl_z, pipe_x, pipe_y, pipe_z. (Derived canonical form). cyl_z and pipe_z are the same as cyl and pipe, respectively. _x means that the local z-axis is directed to the x direction. Hence the config file

```
-------------------------------------
1 pipe Pb 0 0 0 / 0 0 0 1 2 5
2 pipe_x Fe 0 0 0 / 2 0 0 1 2 5
3 pipe_y Al 0 0 0 / 0 2 0 1 2 5
--------------------------------------
```

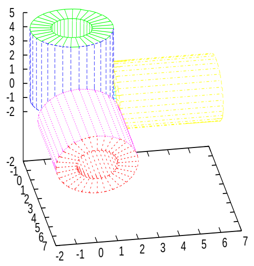


Figure 11: pipe_z(blue-green), pipe_x(purple-red) and pipe_y(yellow)

will have the configuration as in Fig.11

In the case of prism, the 90-degree rotation has more freedom than the case of cyl or pipe. Hence, we denote them by prism_xy (=prism), prism_yx, prism_xz, prism_zx, prism_yz, prism_zy. They are shown in Fig.12

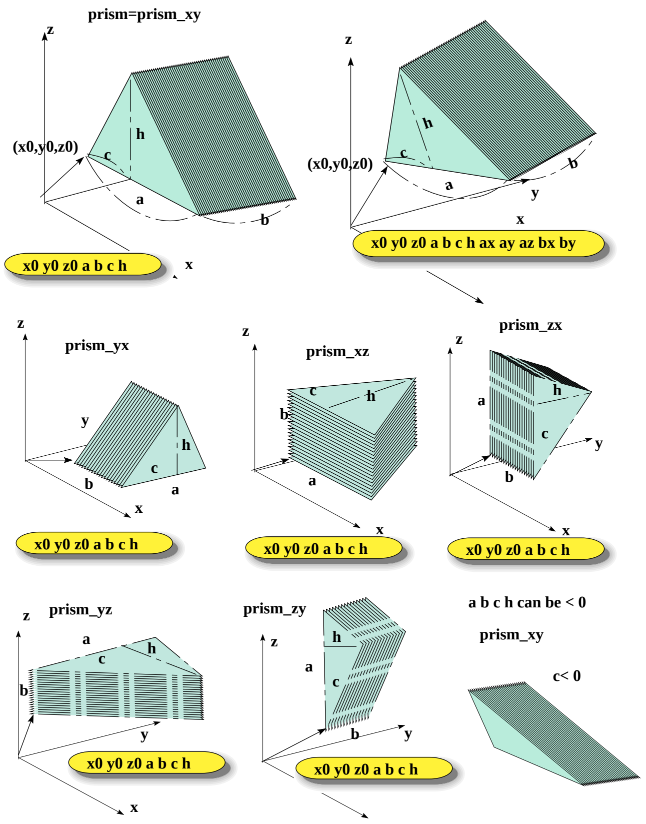


Figure 12: prism and its variants

6.7 The void and world

The void is defined as any position that dose not belong to any of the components in a given detector configuration. Any particle appearing in the void is discarded instantly in the particle tracking.

A world may be defined in a detector or in a subdetctor (For subdetctor, see later). It contains all the rest of components in that (sub)detector. It must be placed last in the (sub)detector definition. The world defined in a subdetector will eventually become an ordinary component. Thus, eventually, only one or zero world will exist in a given detector configuration.

The world may or may not be defined. In some cases, however, it is mandatory. In another case, faster or slower computation would result than the case without the wold.

There are two types of world: Explicit and implicit. The explicit world is made of one of a box, sphere or cylinder. In a configuration file, it is indicated by 'box_w', 'sphere_w' or cyl_w. To define the world by a box, for instance, as the 100-th component, you may give,

100 box_w sp 0 0 0 / 0 0 0

The medium of the world is arbitrary; 'sp' may be most frequent but can be, for instance, 'H2O', if you place detector components in a water pool. The origin and attributes may be given as 0 0 0. Then the system will find a minimum canonical (i.e, non inclined) box with an appropriate origin to contain the all the rest components. (However, partially contained components are disregarded in this calculation; for the partially contained components, see later) You may give non 0 world attributes, if you want to have a larger world than it would be without the specification.

To give the world by a sphere,

100 sphere_w H2O 0 0 0 / 0 0 0

The cylinder world can be defined similarly. Others are the same as the case of the box world.

The implicit world can be made of any volume-shape such as prism, cylinder etc. In this case, you don't specify it, for instance, by prism_w. You have to place the implicit world in the last of the config file and indicate that it contains all the rest of the components yourself. This type of world may be needed in the definition of a sub-detector which will be described later.

If the world were not given in the config file of FirstKiss, a particle exited from a box would have been regarded as in the void; tracking of the particle is ceased there and it cannot enter into another box anymore. Therefore the world is mandatory in this case.

6.8 Mutual relation of components: Overlapping components

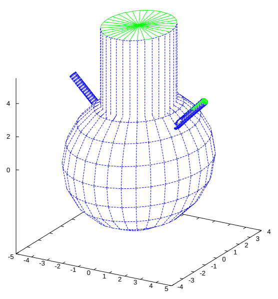


Figure 13: A complex object made from a sphere and 3 cylinders. They are all made of Pb and thus have sense.

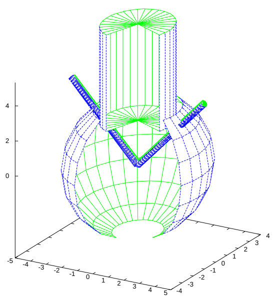


Figure 14: Internal structure of the left picture.

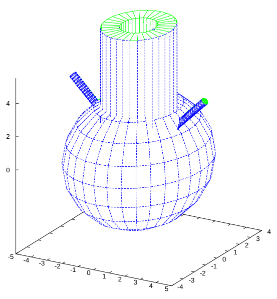


Figure 15: If the central cylinder is replaced by a pipe, the object becomes unusable without further specification.

What happens if two volumes overlap each other ? The rules for mutual geometrical relation of volumes are described here. It is worth remembering that it is very difficult to judge geometrical relation (a component includes another etc) by a computer program, Epics dose not try to do so and all the relations must be specified in the config file by the user. If a wrong specification is given in the config file, the result you get will be odd although the system might not issue any warning messages. You have to carefully examine the configuration of the components by using 'drawconfig'.

- Two or more volumes overlap while the user dose not give a specification that one contains the other. This case has meaning only if the media of these volumes are the same. If they are not homogeneous, the result is unpredictable. Using this feature, you may make a new volume-shape like in Fig.13. Note, however, if a particle is in the overlapping region, the particle may be judged as in one volume in one case and the other in another case.

  If the central cylinder is replaced by a pipe (Fig.15), the configuration becomes conflicting because the central part of the pipe is void (actually not a part of the pipe!); the void and Pb are mixed in the sphere. As naturally understood, a world is necessary to use a configuration like in Fig.13

  The config file used for Fig.13 is

  ```
  ------------------------------------
  #eq  cos45      .7071067
  #eq  mcos45   -.7071067
  1 sphere Pb 0 0 0 / 0 0 0  4
  2 cyl   Pb 0 0 0 / 0 0 2.5  2 5.5
  3 cyl   Pb 0 0 0 / 0 0 0  0.2 6  cos45  cos45
  4 cyl   Pb 0 0 0 / 0 0 0  0.2 6  mcos45 cos45
  ------------------------------------
  ```

6.9 Mutual relation of components: Matryoshka

- It is sometimes necessary to have a configuration that one volume contains another which in turn may still contain the other. and so on, like a famous Russian doll, 'matryoshka'. We express this situation as contain/contained relation, or use a term matryoshka, or mother/daughter relation (the latter is used in 'drawconfig'). **The material of a given point is regarded as that of the inner most matryoshka to which the point belongs.**

- The explicit world is an example of matryoshka. For other cases, the user must specify the relation explicitly in the config file. Suppose a cylinder contains two boxes, and one of the boxes contains a sphere. Then, the file would look like

  ```
   ---------------------------
  1  cyl  Air  0 0 0 / 0 0 0 5 10 / 2  3
  2  box  ...         /  ..
  3  box  ...         /  ..               / 4
  4 sphere  ..      /  ..
  ---------------------------------
  ```

  Note that the user must not specify that the cylinder contains 4th component 'sphere' since it is indirectly contained by the 3rd component 'box'.

- What happens if, for example, the 3rd component is not actually contained by the cylinder ? For a particle in the cylinder, Epics tests whether the particle is inside the 3rd component; this fails always since the box is actually not contained by the cylinder, resulting in a waste of computing time. Since the 3rd component is assumed to be in the cylinder, Epics also fails to find it even if a particle is in the 3rd component (stealth).

- What happens if, for example, the 3rd component is not fully contained by the cylinder ? The uncontained part will become stealth in one case and existent in another case, thus, leading to unpredictable results.
- What happens, if a component is actually contained by another component, but the user forgets to give matryoshka relation in the config file. The contained component becomes (partly or completely; it depends) stealth and the user cannot get a correct result.

6.10 Mutual relation of components: 'partially contain/contained' or 'clipping'

To enable a sort of clipping of a volume by another volume, 'partially contain/contained' relation is available.

An example is seen in *Epics*/UserHook/Clip; suppose we want to make a configuration like Fig.16, that is, a hemi-sphere water above which box shape detectors are placed (don't worry about a water vessel here), which in turn are contained in a vacuum cylindrical tank made of Fe.

The config file is

```
#  type  matter  c    de maxpath   / x y z  a  b c
-------------------------------------------------------------------
1   sphere  H2O 0  0        / 0  0  50  50.
2   box  sp    0  0       /   -50 -50  0  100 100 50 /  -1
3   cyl  Fe    0  0    0 /     0  0  50  50 0.1
4   cyl  Fe    0  0    0  /    0  0  +   50 50  / 5
5   cyl  sp    0  0    0  /    0  0  =   49 = /  6 7 8 9 10 11 12
6   box  pb    0  0      /   -20  -20  55   40 40 .5
7   box  scin 0  0     /    =  =       +     =   =   5
8   box  sp    0  0     /    =  =        +     =   =  10
9   box  pb    0  0      /   =  =       +     =  =    .5
10   box  scin 0  0      /    =  =     +     =  =   5
11  box  pb    0  0      /   =  =      +     =  =   .5
12  box  scin 0  0     /    = =      +     =  =   10
-------------------------------------------------
```

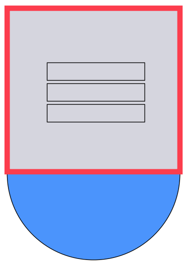


Figure 16: Schematic view of the detector

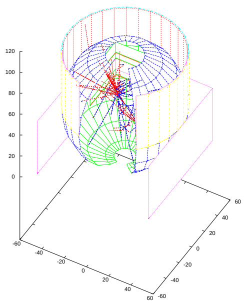


Figure 17: Default display with tracks

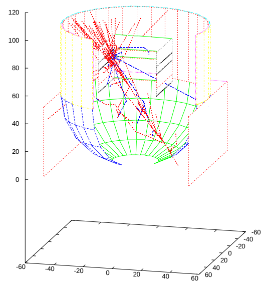


Figure 18: upper hemisphere is made stealth

The first component 'sphere' is partially contained by the second component 'box'; the partial containment is indicated by '/ -1' in the second line. Then the uncontained part of the sphere by the box becomes stealth (sterile). In this case, the sphere becomes a hemi-sphere; the upper half hemi-sphere becomes stealth[12]. Figure 17 shows default display of the configuration with cascade tracks. Since 'drawconfig' cannot clip the sphere automatically, we set the angular range of the

[12]A hemisphere could actually easily be realized by using a 'cut sphere' which is a new volume-shape implemented in *Epics*/prog/NewVol. See later for the description of additional volumes.

sphere by drawconfig to obtain Fig.18. We see that the upper hemi-sphere has no effect on the cascade development.

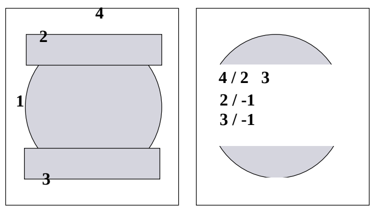


Figure 19: One sphere may be partially contained by 2 'sp' box (left) to form two caps (right). We would need a larger component containing these two (maybe world).

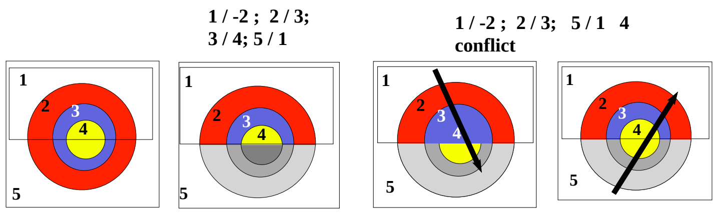


Figure 20: Suppose a configuration on the left. If the 'contain/contained' relation is like 2nd one, clipping applies to all the descendents ; gray parts become stealth. If the relation has conflict like 3rd/4th ones (4 is not in 3 but in 5), the configuration would look like 3rd one for a down going particle but the 4th for up going one.

One volume may be partially contained by more than two objects (Fig.19, 21). It is important to note that 'partial contain/contained' relation is not equivalent to 'clipping'. In the example above, if the box were made of Pb but not of 'sp', cascade could develop in the lead part before entering into the water part.

Figure 20 illustrates what happens when a matryoshka is partially contained. If a component B is partially contained by A, and B contains C which in turn contains D,..., then clipping will be applied to all the descendent components of B. If there is a conflict in the relation such as last two in Fig.20, the configuration would look like different depending on the path of a track, and simulation would give a wrong result.

```
-----------------------------------
1  sphere Pb   0 0 0 / 0 0 0 3
2  box   sp    0 0 0 / -3 -3 2  6 6 1  / -1
3  box   sp    0 0 0 / -3 -3 -3  6 6 1 / -1
4  box_w sp  0 0 0 / 0 0 0
--------------------------------
```

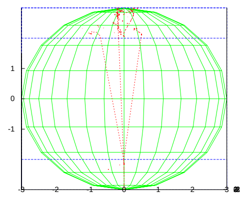


Figure 21: Configuration shown left makes the middle part of the sphere sterile (right); here only electrons are drawn but they go straight in the middle part which would be Pb if not clipped out.

A simple example of partially contained matryoshka is given in Fig.22 together with cascade made by an electron injected from the bottom. The cascade does not develop in the sterile part of the spheres which are clipped out by a box. More realistic example will be given later in the subdetector section.

```
1 sphere Pb 0 0 0 / 0 0 0  1
2 sphere Fe 0 0 0 / 0 0 0  2 / 1
3 sphere Pb 0 0 0 / 0 0 0  3 / 2
4 box    sp 0 0 0 / -3 -3 0  6 6 3 / -3
5 box_w  sp 0 0 0 / 0 0 0
-------------------------------
```

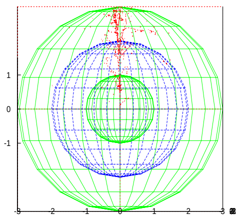


Figure 22: Configuration shown left is to clip 3 hierarchical spheres. Clipping by component 4 is explicitly indicated for component 3 only, but it applies to all the daughters of 3. In the figure, electron tracks are drawn but they go straight in the sterile part of the spheres.

6.11 A possible common mistake with 'pipe'

As mentioned earlier, the central part of a pipe is void, i.e., not a part of the pipe. Therefore, an object placed inside a pipe cannot be contained by the pipe (Fig.23). When a particle emerges in the void, it is immediately discarded in particle tracking. If you want to follow such a particle, the void must be enclosed by a larger component (may be made of 'sp'). Another method for avoiding such a mistake with the pipe is to use two coaxial cylinders; the larger one should contain the smaller one in this case.

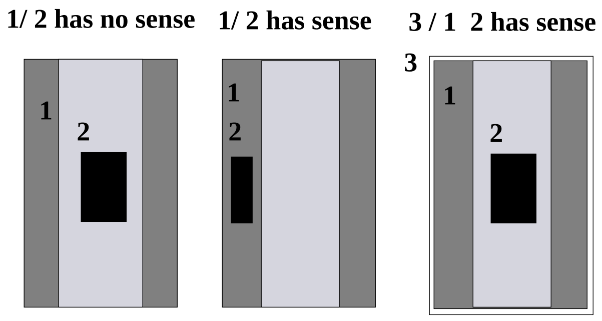


Figure 23: Object inside a pipe cannot be contained by the pipe (left). A pipe can contain objects like the middle figure. To make the first one sensible, a 'sp' box or cyl ay be placed to contain the central object (right) to contain the object.

7 Defining subdetectors

Even for a pile type simple detector configuration, we normally find repeated structures. In such a case, we may define a subdetector and use it as many times as we want.

- To define a subdetector, we must give a name to it.
  - The name must be in lower case characters.
  - It must be within 8 characters.
  - Its first part must not coincide with an existing name such as 'box', 'pipe' etc. That is, box, boxskew, pipe, pipeband etc are not allowed, while skewbox, bandpipe are usable.
- A subdetector must be defined before it is referred to.
- *Epics*/UserHook/SciFiCal/config is an example and shown below.
  - There are 3 subdetector definitions.

- The definition of a subdetector takes the following form:

  ```
  #subd  'name'
  ...
  ...
  #end 'name'
  ```

  where 'name' is the subdetector name.
- As in this example, in the definition of a subdetector (withpb), you may refer another subdetector already defined (belt).
- When referring to a subdetector, the 'matter' field may normally be 'sp' followed by 3 zeros and '/'. Then the origin follows it.
- The origin of a referred subdetector is mapped to this origin.
- If matter field is 'sp', the original media used in the definition of the subdetector are kept in the embedded subdetector. If the 'matter' field has a different medium than 'sp', all the media of the embedded subdetector will be replaced by this medium.
- Optional direction cosines may be added if the embeded subdetector is to be rotated. The subdetector is treated as if it were a box. An example will be shown later.
- These subdetectors are used in the 4th block to define a actual detector.
- This example, when expanded, has 82 components.
- If a subdetector is a matryoshka of another subdetector or of the world, and if the number of components in that subdetector is not 1, we must place a world at the end of the subdetector. The present example is not the case.

```
#  type  matter  c    de maxpath / x0   y0   z0   a  b  c
-------------------------------------------------------------------
#eq  a 60
#eq  b 60
#eq  pbth 0.112
#eq  bgoth 2.5
#eq  scifith 0.1
#eq  scith  1.0

#subd belt
c       x
1  box  scin 0  2  0 /   0   0   0  a  b  scifith
c       y
2  box  scin 0  2  0 /   0   0   +  =  =  =
#end belt

#subd withpb
1  box    pb 0  0  0 /   0   0   0  a  b  pbth
2  belt sp 0 0  0 /    0   0   +
#end withpb

#subd bgocal
c   x
1 box bgo  0  2  0 /  0     0    0  a b bgoth
c   y
2 box bgo  0  2  0 /  =     =    +  =  =  =
c   x
3 box bgo  0  2  0 /  =     =    +  =  =  =
c   y
```

```
4 box bgo  0  2  0 /  =     =    +  =  =  =
#end bgocal



c    top trigger scintillator
1  box scin  0  2  0 /   0   0   0  a  b  scith
c    fiber xy 0
2  belt sp  0 0 0 /    0   0   +
3  withpb sp  0 0 0 /    0   0   +
c    fiber xy 2 + Pb 0.4 r.l
4  withpb sp  0 0 0 /    0   0   +
c    fiber xy 3 + Pb 0.6 r.l
5  withpb sp  0 0 0 /    0   0   +
c    fiber xy 4 + Pb 0.8 r.l
6  withpb sp  0 0 0 /    0   0   +
c    fiber xy 5 + Pb 1.0 r.l
7  withpb sp  0 0 0 /    0   0   +
c    fiber xy 6 + Pb 1.2 r.l
...
...
...
11  withpb sp  0 0 0 /    0   0   +
c    fiber xy 10 + Pb 2.0 r.l
12  withpb sp  0 0 0 /    0   0   +
c  *******  second trigger scintillator *******
13 box scin  0  1  0 /   0   0   +   =  = scith
c    fiber xy 11 + Pb  3 r.l
14  withpb sp  0 0 0 /    0   0   +
...
...
18  withpb sp  0 0 0 /    0   0   +
c    fiber xy 16 + Pb  9 r.l
19  withpb sp  0 0 0 /    0   0   +
c    fiber xy 17 + Pb  11 r.l
20  withpb sp  0 0 0 /    0   0   +
c    fiber xy 18 + Pb  13 r.l
21  withpb sp  0 0 0 /    0   0   +
c    BGO cal.
22  bgocal  sp 0 0 0 / 0  0  +
23  bgocal  sp 0 0 0 / 0  0  +
24  bgocal  sp 0 0 0 / 0  0  +
25  bgocal  sp 0 0 0 / 0  0  +
26  bgocal  sp 0 0 0 / 0  0  +
27  bgocal  sp 0 0 0 / 0  0  +
---------------------------------
```

7.1 Subdetector, implicit world and partially contained components

When a detector has the world, each sub detector must have a world, if it is composed of more than 1 components. The world may be an explicit or implicit one. An implicit world can be compact and may be used when the world should not overlap with other worlds in different subdetectors. An implicit world also may be made when the last component in a subdetector (partially) contains other components.

If we want to make an object like in Fig.24 (upper left), we may make a subdetector consisting of a cylinder which partially contains other 6 smaller cylinders made of 'sp' (lower fig. is the side view of such a configuration). In this case, the bigger cylinder is not 'sp' (say, 'Fe') and smaller 6 are 'sp'. If we interchange 'sp' and 'Fe', the resultant configuration becomes an object shown upper right in the fig. Cascade tracks shown in Fig.25 will confirm that which part is space. This config file is in *Epics*/UserHook/PCtest.

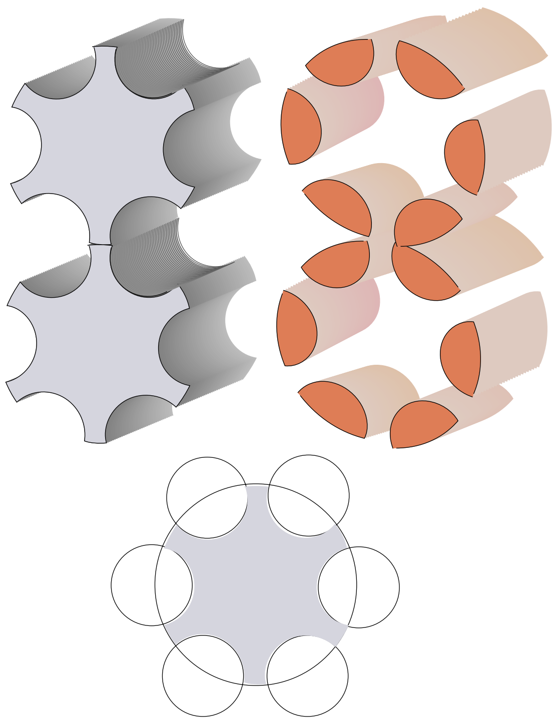


Figure 24: Gear like objects. Upper left and right objects can be made from the same configuration (below); the difference is only in which of the components are 'sp'.

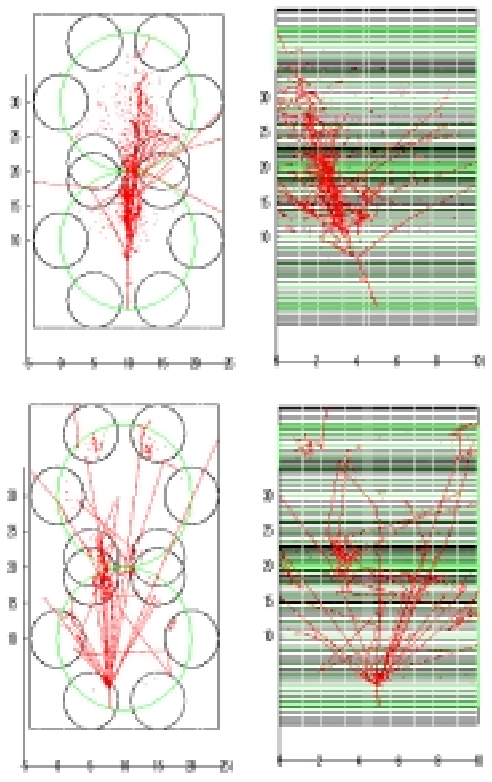


Figure 25: Charged particle tracks in a cascade shower developed in the configuration show left(x and y image). Upper one corresponds to the left object and the lower one to the right.

8 Userhook

Although more words are necessary for how to describe detectors (see Sec.9 for further details), we here describe the user interface (we call 'userhook') to make an desired application. The userhook is managed by ephook.f in UserHook/TemplateIt is better to copy all of the stuff there into a new directory under UserHook, and modify ephook.f for your particular application.

In normal cases, you would simply modify the following parts of ephook.f:

ui1ev : subroutine ui1ev

This is called when all the system level initialization for an event has been ended (except for making the incident particle). Do your own initialization for the event.

You may output some header information for each event if you want to distinguish one event data from another.

userde : subroutine userde(info, aTrack, Move, media)

integer info : input. If info= 0, the particle is still active. If info= 1, the particle dies at B (see Fig.26).

record /epTrack/ aTrack : input. Track information at the top of a segment as shown in Fig.26.

record /move/ Move : input/output. Gives information at the end of the segment as shown in Fig.26. Also it contains information related to the path AB.

record /media/ media : input. Gives current media information.

This is called when a charged particle loses energy in a component for which you have specified to do energy loss counting (in this case info may be 0 or 1). Or when photon or other neutral particle energy becomes lower than a given minimum energy (info =1). It's your decision whether you discard such a particle or count its kinetic energy as energy loss. You can also count up track length.

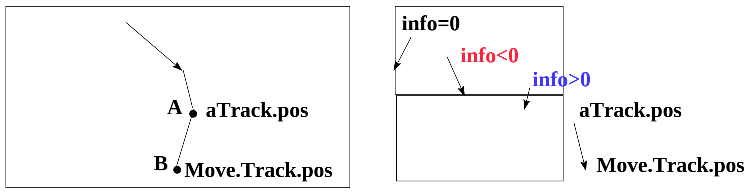


Figure 26: Left: When a charged particle moves from A to B in a component, energy loss information in that segment is given to 'userde'. aTrack and Move.Track are the track information at A and B, respectively. Move.dEeff is the effective energy loss in that segment. Right: When a track is going to cross a boundary of a component, aTrack gives information at the starting point of an arrow and Move.Track the one at the end point of the arrow. If info = 0, the track is exiting to the void, if info < 0, the track is existing to another component, and if info> 0, the track is just entering to a new component. The track position at the boundary is controlled by 'EpsLeng' in the Epicsfile so that the position is apart from the real boundary by EpsLeng along its direction.

There are three types of energy loss: Move.dE, **Move.dEeff** and Move.dEioni (GeV). Move.dE is the true energy loss; it includes ionization loss and possible other losses such as muon's low energy bremsstrahlung. Move.dEioni is the true energy loss due to ionization only.

Normally, energy loss is converted into electric signals by some means or other in the detector and is expressed in terms of the corresponding signals by minimum ionization particles (e.g., relativistic muons). Usually the signal is (can be) assumed to be proportional to the energy loss. However, the photon emission efficiency in organic scintillators shows saturation for heavy ions or slow single charged particles. **Move.dEeff** takes such an effect into account so that the user have to use this for normal applications (Move.dEeff $\leq$ Move.dE).

If you want to interrupt the event generation, return non zero value to the "Move.abort":

- Move.abort = 0 if you want to continue the simulation of this event normally.
- Move.abort = 1 if you want to abort the generation of this event, but want to execute ue1ev.
- Move.abort = 2 if you want to abort the generation of this event, and skip ue1ev. (The event number is updated).
- Move.abort= 3 if you want to discard the particle but continue to simulate the event.

If you are not interested in an event whose first interaction point is deep in the detector, you may give Move.abort 2 to discard the event.

The return value, Move.abort=3, may be used for a particle for which Epics stops its tracking but continue the simulation popping up the next particle from the stack. This may be used when you use some formula for further development of cascade shower without using the Monte-Carlo method.

If the user wants to generate **Čerenkov radiation**, 'userbd' is the place for that.

userbd : subroutine userbd(info, aTrack, Move, media)

integer info : input. See Fig.26

record /epTrack/aTrack : input. See Fig.26

record /move/ Move : input/output. See.26

record /media/media : input. If info ≤ 0, media information at the current position. If info > 0, media information at the new component.

This routine is called whenever a particle crosses a detector component for which you specified particle number counting (1, 2 or 3). Move.abort has the same meaning as in in the "userde" routine. This is also the place to generate transition radiation, if you want.

ue1ev : This is called when one event generation is ended.

Figure 27 illustrates all userhooks with higher hierarchical Epics routines.
Further details of the userhook can be found in ephook.f.

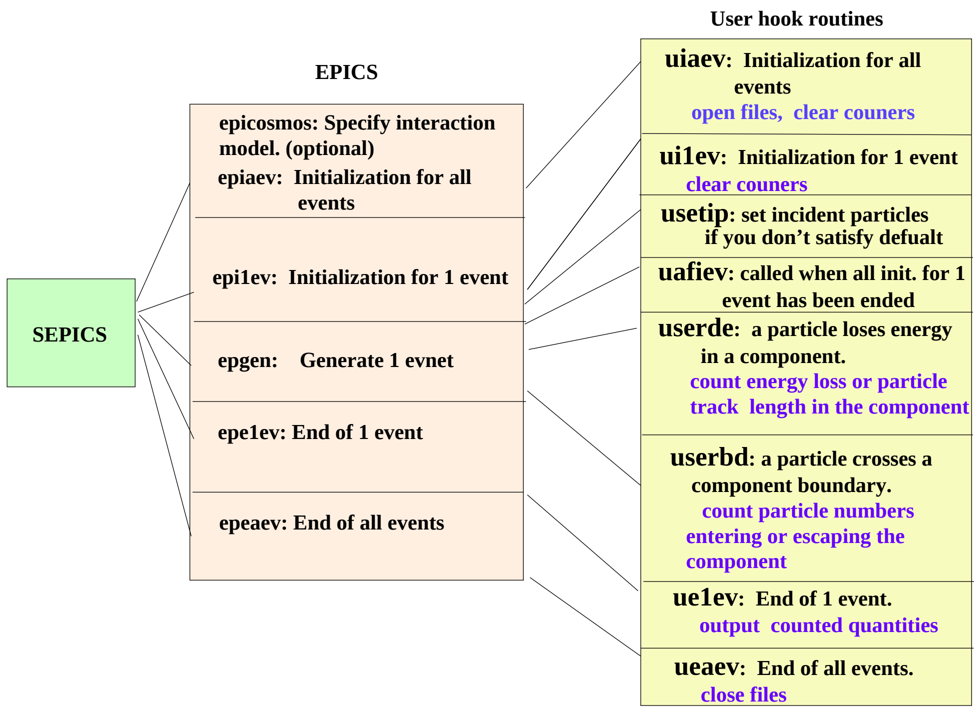

User hook routines
EPICS
SEPICS
epicosmos: Specify interaction model. (optional)
epiaev: Initialization for all events
epi1ev: Initialization for 1 event
epgen: Generate 1 evnet
epe1ev: End of 1 event
epeaev: End of all events
uiaev: Initialization for all events
open files, clear couners
ui1ev: Initialization for 1 event
clear couners
usetip: set incident particles if you don't satisfy defualt
uafiev: called when all init. for 1 event has been ended
userde: a particle loses energy in a component.
count energy loss or particle track length in the component
userbd: a particle crosses a component boundary.
count particle numbers entering or escaping the component
ue1ev: End of 1 event.
output counted quantities
ueaev: End of all events.
close files

Figure 27: All the userhook routines. Blue text is a typical business you may do.

9 Examples

9.1 FirstKiss

#eq in the config file has been explained. Some boxes are inclined so that the direction cosines of edge 'a' and 'b' must be given. They follow the 'a b c' attributes as shown below.

```
3    box  Pb    0  0    / 0   0   7  =    35.  pbth 1 0 0 0    cos15  cos75
4    box  scin 0  de    / 0   0  =addz1   =  =    scth   1 0 0 0    cos15  cos75
```

In this example, 'a' is not inclined so that '1 0 0' is given while 'b' is inclined and makes 90, 15 and 75 degrees with the x, y and z axes, respectively. Therefore we give '0, cos(15) and cos(75)' after '1 0 0'. The direction cosines of 'c' is automatically determined and we need not give them.

The file 'epicsfile' contains one non default parameter. That is, 'Trace' is specified to be 't' ; the system will output particle track information. The file for that is '/tmp/userid/trace$i' since TraceDir is default (' ') where 'userid' is your login name and $i is the event number. You have to make this directory (/tmp/userid) before you run the job. The file 'sepicsfile' is more often customized, but in this example only 'Xinp' and 'Yinp' are modified. They show the (x,y) position of the incident particle. The incident particle is determined by the file 'primary' (see 'PrimaryFile' in sepicsfile). The 'primary' file in this example specifies that the incident is a proton (total) energy 100 GeV.

9.2 Managing scintillating fibers or segmented media

Scintillating fibers are normally incorporated into a detector as a belt in which a number of thin fibers are laid. The cross section of a fiber is normally a circle or square. Silicon strip detectors are segmented into a number of narrow strips. If we regard each fiber of segment as a separate component, we will have a large number of detector components. You may treat each fiber as a component; you will need almost no programming in ephook.f. However, you may treat such configuration as follows.

There is always some inhomogeneity between fibers or segments, but it can be neglected in most of applications; we may treat such a fiber belt or silicon detector as a homogeneous one. Then each fiber or strip in a layer can be handled in a userhook routine (userde or userbd). (In sections9.6 and 9.7, we will learn how to describe a lot of blocks in a 'config' file)

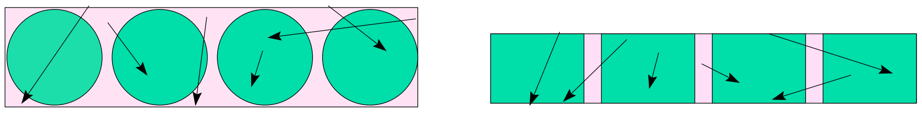


Figure 28: Each arrow shows a straight segment of a zigzag track of a charged particle

We assume that the layer is homogeneous and count, say, energy loss in a fiber, that is the loss in a green region (Fig.28). Although Epics gives the loss by a whole track denoted by an arrow, it is possible to compute the portion (let it be p) of the line contained in the green region; we know the starting position of the track, aTrack.pos, the segment length, Move.dl (cm), and the direction cosine, aTrack.w.

If we give 1 in the "de" part of a configuration file, the energy loss given by Epics, "Move.dEeff", is the average energy loss in the path length "Move.dl". (Move.dE is the true energy loss while Move.dEeff is the effective energy loss in which saturation effect of organic scintillator is taken into account). Therefore we may get the loss in the green part as $p\times$"Move.dEeff". If we give 2 in the "de" part, the energy loss given by Epics, "Move.dEeff" is a sampled one from the energy loss fluctuation distribution of which the average is the same as the case of "1". The energy loss in the green part would be $p\times$"Move.dEeff". However, strictly speaking, this is not a correct treatment if we consider the energy loss fluctuation because it depends on the path length. More accurate treatment is possible by some means. However, the difference is very small so that you may employ the method described here.

The directory *Epics*/UserHook/SciFiCal is an example to manage a number of scintillating fiber tubes. In that ephook.f, square fibers are assumed and *Epics*/UserHook/sqfiber.f is used to count up energy loss in a number of fibers.

9.3 Managing scintillating fibers 2

The method given in the previous section needs somewhat complicated programming to manage a number of virtual fibers, though the detector configuration is simple. In some case, we may need to incorporate a number of fibers directly and treat each fiber as a component. In this case, we don't need programming for managing the fibers. It's not rare we have a number of fibers of order 10^4; we must give the configuration so that time needed to search for the component where a given particle exists be small. To realize such a configuration, we must use subdetectors each of which have some moderate amount of components and a world to envelope them. If a particle position is not in a world, we don't need to search for inner components and thus, particle tracking time is greatly reduced. Such an example is in *Epics*/UserHook/BetsDetail2/.

9.4 Subdetector, its rotation and customizing incident particles

Epics/UserHook/SubDet1 has an example of constructing a detector with 8 subdetectors each of which is a rotated version of one and the same subdetector consisting of a matryoshka.

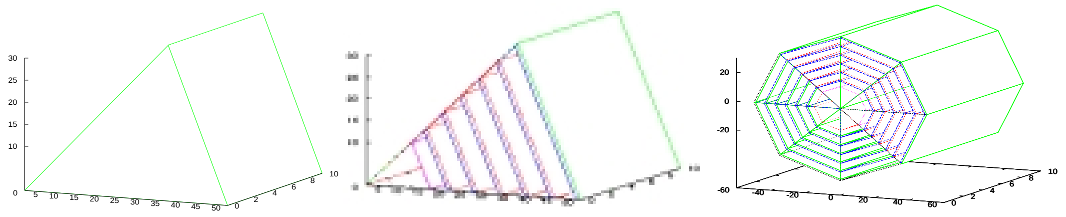


Figure 29: Left: The basic component is an isosceles prism with an apex angle of 45, 67.5 and 67.5 degrees. Middle: Similar components are packed to form a subdetector as a matryoshka. The inner most part is 'sp'. Right: The subdetector is rotated to form a detector as in the list shown below.

```
1 block sp  0 0 0 / 0 0 0
#      rotate it; since "b" is not inclined, b= '0 1 0'
#      can be omitted  in the last part (cos45 0 cos45 0 1 0 is full)
2 block sp  0 0 0 / 0 0 0   cos45 0  cos45
3 block sp  0 0 0 / 0 0 0   0 0 1
4 block sp  0 0 0 / 0 0 0   mcos45 0  cos45
5 block sp  0 0 0 / 0 0 0   -1  0 0
6 block sp  0 0 0 / 0 0 0   mcos45 0  mcos45
7 block sp  0 0 0 / 0 0 0   0 0  -1
8 block sp  0 0 0 / 0 0 0   cos45 0  mcos45
```

A subdetector, block, is incorporated in the first component as it is. The second one is rotated by 45 degrees around the y-axis. cos45 has been defined by #eq (not shown). Remember that, in drawconfig, you can use the subdetector mode to show one selected subdetector.

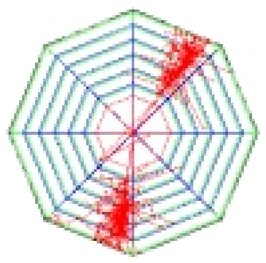


Figure 30: Side view with cascade tracks

In this example, two incident particles are set in 'usetip' of ephook.f. Therefore many of the parameters in 'sepcisfile' is neglected. The primary specified by 'primary' is also not referred to. As the code fragment in 'usetip' shows, to set incident particles, the user may create a track information (particle attribute, 4-momentum, position, direction cosines) and call epputTrack as many times as you want (see p.48). . Finally, you set a flag (icon) to indicate that you have set the incident particles.

9.5 Using Cosmos interaction routines directly

As we have seen in the previous section, in 'usetip' we can define incident particles. A more elaborated example is shown in '*Epics*/UserHook/LHC where a simulation of an experiment at a collider is assumed; observation of high energy particles inside the beam pipe where secondary particles generated by interactions of particles with the beam pipe become a big background. To generate particles at the intersection, p-p collisions is managed by employing a Cosmos function.

The beam pipe length and diameter in the example are only for showing how collider interaction occurs; a longer length and smaller diameter would be needed for a realistic simulation.

Since the inner part of a pipe is void, an implicit world, 'sp' cylinder, is assumed to contain all the rest components.

- To control p-p interactions, some parameters not listed in the 'sepicsfile' or 'epicsfile' must be introduced. They are given in the file 'other' ; the file is opened at 'uiaev' (initialization for all events) and the parameters are read there and shared with other subprograms through the named common specified by Zother.h. Alternative way is to use the method in section16.
- Two colliding protons are made also in 'uiaev'.
- In 'usetip', the p-p collision routine 'chncol' is called via 'ephookPPCol'.
- Each produced particle is made as a track and put into the stack by epputTrack (see p.48).
- Particles are observed in 'userbd'.

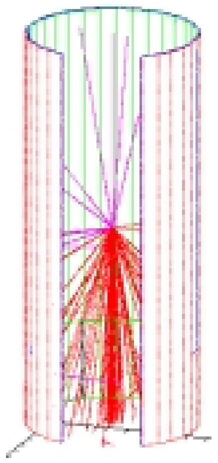


Figure 31: Quasi-single diffractive event at an LHC energy

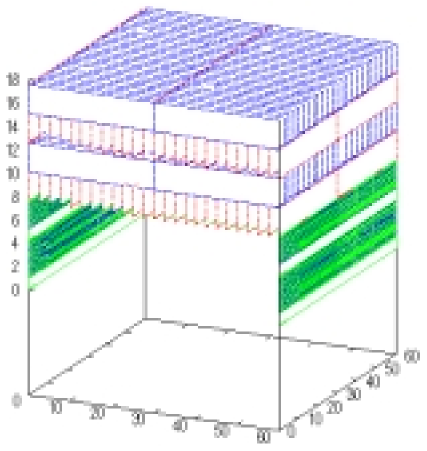


Figure 32: Upper structure shows a number of BGO blocks.

9.6 Incorporating a number of blocks

In 9.2 we managed a number of BGO blocks by assuming a homogeneous slab; each BGO block is virtually constructed in the slab within ephook.f. This made the configuration description simple but the we needed a bit complicated programming. We can take inverse approach by putting a number of blocks in the configuration and count energy loss in each block with a simple program.

The directory SubDet2 contains an example of such a configuration. We define a subdetector (bgocal) consisting of 4 blocks of BGO. Then, gathering 6 of such a subdetector, we make a larger subdetector (bgocalx). By rotating bgocalx, we make a similar subdetector for the y measurement (bgocaly). Finally, a number of bgocalx and bgocaly is incorporated in the main detector.

9.7 More realistic configuration and a world in a subdetector

The configuration in the previous section introduced how to set up a number of blocks. However, each adjacent block has no gap inbetween while the realistic configuration would have some gap between the blocks. SubDet3/config shows such a realistic configuration in which periodic structure is observed. As an example, each block is assumed to be surrounded by Al of thickness 1 mm.

To start with, we define a subdetector, bgox (Fig.33); 4 BGO blocks are embedded in a explicit world made of Al. **A subdetector may have one world** as in this example. When it is included in a higher hierarchical construct, the explicit world is changed to be a simple matryoshka. As a result, each BGO block wears a 1 mm thick Al armor. Such a unit is integrated into a larger unit, 'bgocalx', two of which in tern make one slab with x position resolution, 'planex', . Then, 'planex' is rotated by 90 degrees to form a slab with y position resolution, 'planey'. Finally, 'planex' and 'planey' are piled up to obtain a calorimeter. In the config file, you see a notation, "+m2" in the x column ("m2" is defined by #eq to be 0.2). This is to give an offset of 0.2 cm to a value which would result if only "+" were given, and thus affords a gap of 0.2 cm between adjacent components.

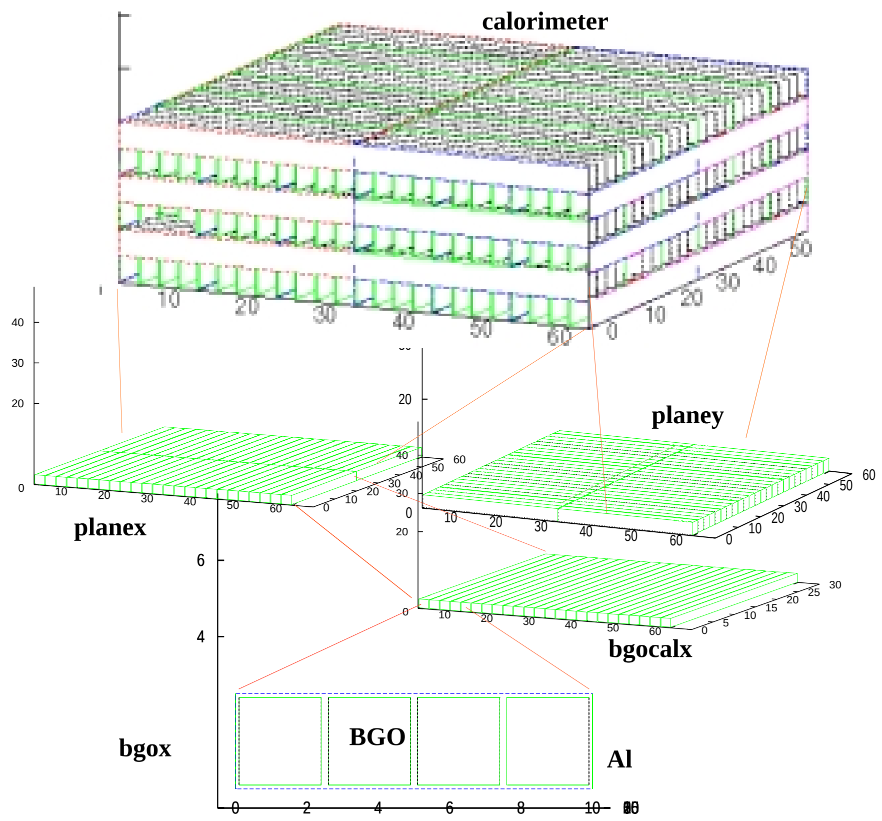


Figure 33: Constructing inhomogeneous periodic configuration

9.8 The “=v” notation

As an example of using the “=v” notation in the x (y or z)-origin column (v is a numeric value or a symbol defined by #eq), Shift/config is prepared. The difference of “=v” from “+v” (in the case of the x-origin column) is that the latter add v to (x-width value + x-origin value of the previous component) while “=v” add v to x-origin value of the previous component. Using this notation, we can easily give a configuration shown in Fig.34.

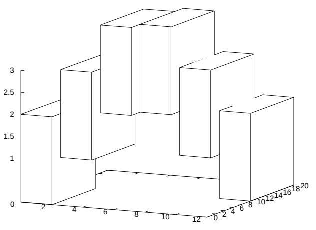


Figure 34: The “=v” notation helps make configuration like this

9.9 Muon transport

In the transport of charged particles, the ionization loss is always considered (but see the 'Knckon' parameter). In the case of muons, energy loss by the direct pair creation, bremsstrahlung and nuclear interaction become important at high energies (normally at few 10 GeV for high (Z,A) material, and few 100 GeV for low (Z,A)). Independently of the parameter values which control these 3 processes, if the loss due to each of the 3 processes is <1 % of dE/dx (ionization), the corresponding process is neglected. Also, if a given media file has no table for the muon energy loss, the loss is neglected. Among the 3 processes, nuclear interaction has least significance, then follows bremsstrahlung. The direct pair creation has the largest contribution which mainly comes from a number of low energy pair emissions. However, the nuclear interaction and bremsstrahlung may sometimes give a large loss and lead to big fluctuation.

The 3 parameters, MuNI (nuclear interaction), MuPr (pair creation) and MuBr (bremsstrahlung) are in the 'epicsfile'. Each of these may take an integer of 0 to 3. If 0 is given, the corresponding interaction is completely neglected. For 1, the energy loss by the corresponding interaction is included in dE/dx of muons as a continuous energy loss. Let $v = E_{transfer}/E_\mu$, the loss here is

$$\int_{v_c}^{v_{max}} E_\mu v \frac{d\sigma}{dv} dv \qquad (1)$$

where $v_c \sim 0, v_{max} \sim 1$.

If 2 is given, energy loss by the corresponding interaction is included in dE/dx of muons as a continuous energy loss. However, only the energy transfer smaller than v_{min} of the muon energy is included. (v_{min} is fixed when the user makes a muon table. Normally it will be $10^{-2} \sim 10^{-4}$). The loss here is

$$\int_{v_c}^{v_{min}} E_\mu v \frac{d\sigma}{dv} dv \qquad (2)$$

The portion of loss by $v > v_{min}$ is treated as a stochastic process. However, the product from the corresponding interaction is neglected. The case of 3 is treated similar to 2 but the corresponding interaction is really made to happen.

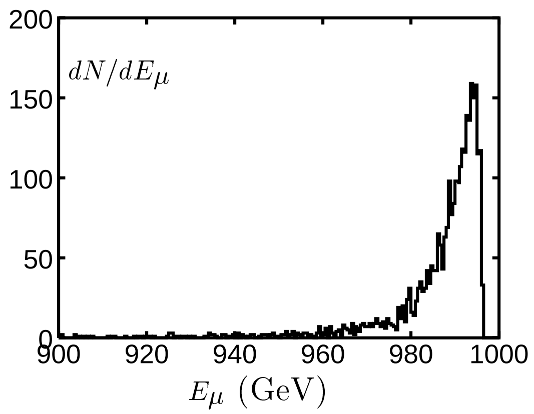


Figure 35: The muon energy distribution after traversing a 3 m slab of iron. The incident energy is 1 TeV.

Tips: If you are dealing with muon transport, say, in rock, perhaps you don't need to follow electro-magnetic cascade. It is better to put high minimum energy for electrons and photons (EminEelc and EminGamma), or to prohibit production of knock-on electrons by making Knckon 'f' (see epicsfile). It will greatly reduce the computation time.

9.10 Magnetic field

If you give MagField = 1, a constant magnetic field can be specified by giving the 3 components of the field to (Bxu, Byu, Bzu) in Tesla (see epicsfile). If MagField = 2, non uniform field is assumed and the system expects that you have made a program unit which gives the 3 components as a function of the particle position. The template of the program unit is eppos2B.f in each userhook directory. It should look like

```
      subroutine eppos2B(aTrack, B)
#include "ZepTrack.h"
      record /epTrack/ aTrack  ! input. charged particle.
      record /epDirec/ B       ! output. magnetic strength in Tesla.
                               !        (B.x, B.y, B.z) must be given in
                               !   the local coordinate of the
                               !   current component.
       end
```

The local particle position is

`(aTrack.pos.x,aTrack.pos.y,aTrack.pos.z)`

which is in a component with component number, aTrack.cn. If you want to convert it to the world coordinate, you may use

`call epl2w(aTrack.cn, aTrack.pos, posw)`

where posw is a record /epPos/ and must be declared beforehand.

`(B.x, B.y, B.z)`

must be given in the local coordinate system. If you know B in the world coordinate (let it be Bw), you can translate it into the local coordinate value by

`call epw2ld(aTrack.cn, Bw, B)`

The constant magnetic field case is in UserHook/MagF.

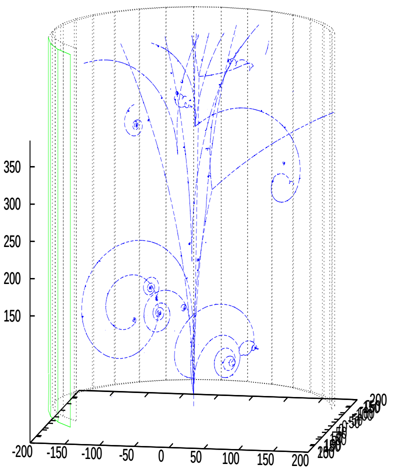


Figure 36: A 50 GeV p-p interaction in liquid hydrogen with a 1 T field

9.10.1 Note on synchrotron radiation and magnetic pair creation

The synchrotron radiation in "normal" circumstances is completely negligible; emitted photon energy is much lower than $\sim$ 100 keV being the standard minimum energy of Epics. However, Epics can treat the radiation if it undertakes in vacuum ('sp') by putting the minimum photon energy very low. The problem in this case is that the electron can make a semi-infinite (in the sense of computer time) cyclotron loop because the energy loss is very small.

There are two relevant parameters in 'epicsfile': Sync and SyncLoop. If Sync = 0, energy loss by synchrotron radiation is not considered, If 1, dE/dx by synchrotron radiation is taken into account for electrons as a continuous loss, though this loss is normally completely negligible. If 2 and if the radiation length is >10 km (normally 'sp'), explicit generation of synchrotron photon is tried (at present, photon angular spread is neglected).

To avoid the semi-infinite loop, if an electron runs more than $(cyclotron\ radius) \times (SyncLoop)$ in 'sp', it is discarded.

A relevant physical parameter for synchrotron radiation is $\Upsilon = \gamma B/B_c$, where γ is the electron gamma factor and B_c the critical field value ($\sim 4.3 \times 10^{13}$ gauss), B the magnetic field. Epics employs the best known formulas (in Cosmos) valid for $\Upsilon \approx 0$ to $\Upsilon \sim 10$. Magnetic pair production is controlled by MagPair. If it is 0, the pair production is neglected. If 1, it is considered. However, the process is always completely negligible until $\chi = E_\gamma/m_e \times B/B_c/2$ becomes order of unity.

10 Making Cosmos generated particles incident

Suppose you want to make a simulation of cascade showers in a detector, using particles generated by Cosmos as incident particles. In this case, you probably read a file to get a (group of) particle(s) generated by Cosmos and input to the particle to Epics. To do so,

- You may open the file at 'uiaev'
- You may read particle data in "usetip" and establish particle energy, angle, particle code, time, incident position on the detector and make a track to call 'epputTrack' (see p.48). The code fragment in ephook.f in LHC or SubDet1 will help this.

  In this case, you have to give "icon = 0" to signal that you have set the incident particle and make Sepics detour the setup of the incident particle.

10.1 Caution

It will take a very long time if you simulate very high energy or very large number of showers in a, say, 10 r.l deep detector. This means that if you use Cosmos generated particles as input to Epics to see the detector response, you need a ultra-super computer. To avoid such situation, it is recommended to make a table of detector response (which can account for fluctuation) for a wide energy range by using Epics.

11 Controlling Epics yourself

Sepics offers rather versatile applications. However, in some case, you may want to write you own main program and call Epics directly for full flexibility. A rough sketch of your program will be as follows.

11.1 Example 1

- In this case, you still need ephook.f. Delete #include "../main.f" in ephook.f
- Modify Makefile to be able to manage your own program (which should include the main program).

- In your own program, initialize Epics by calling

  call epicosmos(param)

  call epiaev(EpicsFile, ConfigFile)

  where param is the parameter file to specify the interaction model, EpicsFile contains the path to 'epicsfile' and ConfigFile that of 'config'. The first call to 'epicosmos' may be omitted if you use the default hadronic interaction code.

- To initialize one event simulation,

  call epi1ev(icon)

  where icon is an integer output. In "usetip", you **may** give incident particles. The value of "icon" here should be 0, if you properly used the "usetip" routine to give the incident. **Instead** , after calling epi1ev, you may "call epputTrack" routine yourself (see p.48).

- To generate a cascade,

  call epgen

- After that, you have to call

  call epe1ev

  to signal the end of 1 event.

- At the end of all the event generation,

  call epeaev

11.2 Example 2

Suppose you are using Cosmos to generate particles and observing them at a given level. If you want to input these particles into a detector, a normal procedure will be to store the data in a disk file first and later you use Epics as explained in a previous section.

However, you can directory simulate the response of the particles in the detector without storing the Cosmos result in the disk (though not recommended in general).

- In this case, you still need ephook.f. Delete #include "../main.f" in ephook.f
- Get chook.f in the Userhook of Cosmos.
- Modify Makefile as in the previous example.
- Add `#include␣Zcmain.h` in chook.f
- In chookBgRun, initialize Epics by calling

  call epicosmos(param)

  call epiaev(EpicsFile, ConfigFile)

- In chookObs,

  call epi1ev(icon)

  and you may establish an incident particle as in the previous example.

  To generate a cascade,

  call epgen

  After that, you have to call

  call epe1ev

  to signal the end of 1 event.

- You may probably need to do something in chookEnEvent to distinguish one group of particles generated by one primary.
- At the end of all the event generation, i.e, in chookEnRun

  call epeaev.

12 Distributing jobs over a number of workstations

This is a similar function as in Cosmos, and employs most of the scripts from Cosmos. *Epics*/DistJob contains relevant stuff. If you have never used the DistJob function in Cosmos, you must first compile echoErr.c in *Epics*/DistJob,
   cc -o echoErr echoErr.c
and put echoErr in DistJob/bin/[13]. In this directory, you have to issue,
   source addpath
to add (*Epics*/DistJob, Cosmos/DistJob)/bin to the command search path. To do a new distributed job, you may copy Template there and give an appropriate name. You may work in that new directory for distributing jobs. The Readme file will tell you how to use the scripts. The scripts, distjob and collect, in 'bin' are different from those given in Cosmos/DistJob/bin.

13 Creating a new material file

You can see the readymade media files in the *Epics*/Data/Media/ directory (see also 18.1). A media file keeps various constants relevant to the material, cross-section tables and random sampling tables. If you cannot find the material you want to employ, you have to create a media file. The media files can be placed in a maximum of 3 different directories, which are specified by MediaDir in 'epicsfile'. If the same material name is given in two or three directories, earlier one has priority. The procedure is as follows:

- Create basic media file by hand as in *Epics*/Data/BaseM/

  As an example, take BGO there:

```
#name   Format    Bi4Ge3O12
   BGO     1
#
#  Elem  rho(g/cm^3)    Gas/Solid(1/0)   refl.index Birks C

     3         7.1              0          2.15     0 0 0
#       Z       A        N
        83      208.98   4
        32      72.81    3
        8       16       12
```

  – You can put comment by putting # in the first column.
  – In the name field, the material name must be given. It is case sensitive (except for media existed in older Epics), and must be within 8 characters that can be a file name. Format should be 1 at present.
  – The 'Elem' field shows the number of elements (integer) composing an unit of the material, 'rho' field the density (g/cm^3), 'Gas/Solid' field 0 for solid or liquid, 1 for gas, 'refractive index' field the refractive index. If it is not known or not used, give 0.

[13] echoErr is used also in *Epics*/Util/Elemag. Therefore the best place to put it would be *Epics*/Scrpt/. However, you should remember that it is not a script and thus machine architecture dependent. Unless the mab (multi architecture binary) is supported by the system, you have to change it when you move to a different architecture machine by some means or other.

For gas, give $(n-1) \times 10^6$. In the 'Birks C' field, give Birk's correction factor if the material is organic scintillator (see R.L.Craun and D.L.Smith. Nucl. Instr. and Meth., 80:239-244,1970.) else 3 zeros.

  - In the Z, A, N fields, atomic number, atomic weight and number of that elements must be given. The number of rows should be equal to 'Elem'. It should be noted that **'N' is not necessarily an integer.**

    For example, in the case of 'Air' (mixture of 78 % N, 21 % O an 1 % Ar in volume), we see the following Z, A, N.

```
#        Z        A        N
         7        14       1.56
         8        16       .42
        18       40       .01
```

    Then, a unit of Air ("virtual molecule") contains 1.56 Nitrogens, 0.42 Oxygens and 0.01 Argons.

- The basic material file may be placed in any directory.
- After completing a base material file, you may go to *Epics*/Util/Elemag
- Issue,

  CreateTab base-material-file media-file-directory

  where the 'base-material-file' is the path to the basic material file you made and 'media-file-directory' a **directory** (say, /tmp) where you want to save the new media file (**the file name becomes the same as the base material file name**). A typical example would be

  CreateTab ../../Data/BaseM/matterX ../../Data/Media

- During the course, you will be asked whether you want to make tables for muons (for bremsstrahlung, pair creation and nuclear interaction). They may not be created depending on the material and purpose. Remember that we can add the muon tables later by issuing

  CreatemuTab base-material-file media-file-directory

  in *Epics*/Util/Mu.

- **Note:**.

  If 'Elem' ($= n$) is large, table making takes rather long time ($\sim n$ times the single element case).

  In most of cross-section treatment, the basic cross-section for each element is added with a appropriate weight to make the cross-section for the material. That is, we don't use effective Z, A by which we approximate a molecule (or compound material) as a single atom as might have been taken in some other codes. This is why we need long time for large n in making the tables.

- One more job is need for dpmjet3, if you use it as a hadronic event generator. It is to create xxx.GLB and xxx.inp for medium 'xxx' as seen in *Epics*/Data/Media. The simplest way would be:
  - Go to *Epics*/Data/Media
  - Confirm that there is no garbage files; if any, delete them. Also confirm that there is 'xxx' and it's basic media file is in *Epics*/Data/BaseM.
  - Issue, 'iniGlauberAll'

    To be able to use this command, 'source Scrpt/sevi' must have been executed at *Epics*. This command makes .GLB and .inp files for all media which have no such files. This takes rather long time. In some case, the process may fail by unknown reason. Then, retry.

- If you want to make .GLB and .inp files for a particular 'xxx' at an arbitrary location, please see *Epics*/Scrpt/iniGlauber.
- .GLB and .inp files in *Epics*/Data/Media are not directly used by Epics. The command, fordpmjet *configfile*, will create new dpmjet.inp and dpmjet.GLB files for a particular *configfile* from .inp and .GLB files found in *Epics*/Data/Media. The dpmjet.inp and dpmjet.GLB files must be placed in a particular application folder.
- Tips. If you have no xxx.GLB and xxx.inp files for media xxx. However, the elements in 'xxx' are found in other media. (Say, suppose 'xxx=CO2'. C is in Carbon, and O is in BGO.) In such a case, you may make a minimum dummy config file which contains Carbon and BGO. (Let it be dummyconfig). Then, you may issue 'fordpmjet dummyconfig'. This will create necessary files.

14 Creating a new volume-shape

If the predefined volume-shapes such as box, sphere etc are not enough for your purpose, nor tricks such as clipping can meet the requirement, you can make your own volume-shape. Although the existing volume-shapes, 'cone', 'cap' and 'ecyl' look like predefined ones, they are examples of new volume-shapes. Therefore, we first describe how to use new volume-shapes.

14.1 How to use new volume-shapes

- Make a config file which contains lines like,

  ```
  #news  new-1 xxxx
  #news  new-3 yyyy
  ...
  ```

  (#news for new shape) as many as you need, where xxxx, yyyy, ... are the name to represent a new volume shape (say, cone; a max of 8 lower case letters).

  You may use any number from 1 to 15 for 'i' in 'new-i'. Once you define xxxx by #news, you can use the name, 'xxxx', same as box, sphere etc.

- Go to *Epics*/Util and issue

  ./usenewvol 'configfile'

  where 'configfile' is the path to the config file mentioned above. This will take care all the necessary things for using new volume-shapes[14]

- For your new volume-shape, you have to make some programs to handle it. Then, you have to compile it to include it in the library.

- In the case of cone, cap and ecyl, the routines that handles these shapes are in *Epics*/prog and has already been compiled to make the library.

- There are more than 20 new volume-shapes defined in prog/NewVol. They are also ready for use as described above. See a separate manual for their details.

[14]epNewVolume.f in *Epics*/prog is updated by the command above. The library is also updated. Also drawconfig is updated.

14.2 How to make a handler for new volume-shape

When you introduce a new volume-shape (let's name it 'xxxx'),

- **For tracking.** You may make a file ep_xxxx.f (you may choose another name). Then, replace one of epdummy?.f in *Epics*/prog/NewVol by it. Replace all epdummy? in Makefile there by ep_xxxx.

  The file should contain the following 5 subroutine subprograms:

  epbxxxx, epsxxxx, epenvlpxxxx, eprxxxx, epatlocxxxx

- **For drawconfig.** To draw the new volume-shape, make a file epDraw_xxxx.f in which you define a subroutine subprogram

  epDraw_xxxx

  and replace one of *Epics*/Util/DrawNewVol/epDraw_dum?.f by that file. Replace all epDraw_dum? in Makefile there by epDraw_xxxx.

14.2.1 A concrete example of the subroutines

As a concrete example of how to make such programs, let's take 'cone' as 'xxxx'. The program file, ep_cone.f, is *Epics*/prog/. It contains the following subprograms. Their essential part is described below.

eprcone To read configuration data for the volume-shape.

```
      subroutine eprcone(comp)
       implicit none
#include "Zep3Vec.h"
#include "Zcnfig.h"
       record /Component/ comp  ! output. to receive the config data.
c             next is mandatory
       call eprpst(comp, 4,  5, 1, 6)
c             you may  check the data below; optional
       end
```

'call eprpst(comp, 4, 5, 1, 6)' is to inform that the volume has 4 attributes (See Table 2 and Fig.8), and the handler needs 5 locations to keep attributes and some computed values for later use. The last 1 and 6 mean that for the non-canonical form, 6 direction cosines are required from comp.direc(1) to comp.direc(6).

Although the mandatory coding is only the lines shown above, it is recommended to append a code fragment which examines the volume attributes read by the system. Volat(comp.vol +i) is the i-th volume attribute. If error is found, you may call cerrorMsg. (See eprcone).

epbcone This will be the most tough part of your coding. The program is to find the length to the boundary from a given point with a give direction.

```
      subroutine epbcone(comp, pos, dir, length, icon)
       implicit none
#include "ZepTrackp.h"
#include "Zep3Vec.h"
#include "Zcnfig.h"
#include "ZepPos.h"
#include "ZepDirec.h"
#include "Zepdebug.h"
```

```
         record /Component/comp  ! input.
         record /epPos/ pos   ! input.  position.
         record /epDirec/ dir  ! input. direction cosine
c      'pos' and 'dir' are given in local coordinate of 'comp'.
         real*8  length !  output length cm from pos to the boundary
         integer icon  ! output 0: length obtained. pos is inside
                      !       1:  //                        outside
                      !      -1: the line dose not cross the volume
```

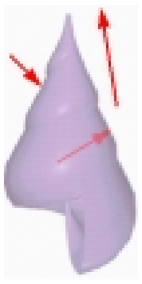


Figure 37: epbxxx must find the length to the volume. Three cases can be supposed.

You can extract the attributes of the volume from Volat(comp.vol +1) etc. (pos.x, pos.y, pos.z) is the position and (dir.x, dir.y, dir.z) the direction cosines; both are local to 'comp'. You have to find 'length' (≥ 0) between 'pos' and the nearest boundary that a half line extending from 'pos' with direction cosines 'dir' crosses this volume. If the crossing point is not found, give icon -1 (black arrow in Fig.37). If the crossing point is found and 'pos' is inside the volume, give icon 0 (blue arrow) else if 'pos' is outside of the volume give 1 (red arrow). If the 'pos' is on the volume surface, regard it as inside.

epscone This is to examine whether a given point is inside a given component and may be used by epbcone. This is relatively easy to make.

```
      subroutine epscone(comp, pos, icon)
      implicit none
#include "Zep3Vec.h"
#include "Zcnfig.h"
#include "ZepPos.h"
      record /Component/ comp !input component
      record /epPos/ pos  ! input. position in  local coord.
      integer icon  ! output. 0--> pos is inside
                    !         1-->        outside
```

The position 'pos' is local to the component, 'comp'. You have to judge whether 'pos' is inside (or on the surface) 'comp' or outside. If inside, give icon 0 else give 1.

epenvlpcone This routine is to return a minimum box that envelops the volume. Easy to make.

```
      subroutine epenvlpcone(comp, org, abc)
      implicit none
#include "Zep3Vec.h"
#include "Zcnfig.h"
#include "ZepPos.h"
      record /Component/ comp  ! input.   component.
      record /epPos/ org       ! output.  origin of the enveloping box
```

```
                                             !          (in local coord.).
      record /ep3Vec/ abc     ! output.  a,b,c of the box
```

epatloccone This is also easy to make. You may establish i-th attribute location in loc(i). Normally loc(i) = i for all attributes. In some case you may want to move the i-th attribute in the input to another location. In such a case, you have to arrange the location; loc(i) should show the real location of the i-th attribute, that is, Volat(com.vol + loc(i)) is the i-th attribute.

```
      subroutine epatloccone(comp, loc)
      implicit none
#include "Zep3Vec.h"
#include "Zcnfig.h"
      record /Component/ comp ! input.
      integer  loc(*)
      integer i
      do i = 1, 5
         loc(i) = i
      enddo
      end
```

14.2.2 A concrete example for a shape drawing subroutine

Above five routines are used when tracking particles. One more routine you have to make is the one that is used when 'drawconfig' displays the volume-shape. The name should be epDraw_xxxx. Our example is thus **epDraw_cone** in *Epics*/Util/epdrawComp.f

```
      subroutine epDraw_cone(comp, p, n)
      implicit none
#include "Zep3Vec.h"
#include "Zcnfig.h"
#include "ZepDraw.h"
      record /Component/ comp  ! input. component
      record /epPos/ p(*)     ! output. (x,y,z) to describe
                              !   cone in local coordinate.

      integer  n               ! output.  number of (x,y,z) data put in p.
```

You have to give coordinates in the form compliant to the gnuplot 3-D display, which is described below:

- Segments of continuous lines are expressed by consecutive rows, in each of which (x,y,z) are given. For example,

  ```
  -34.7296  196.962  .000000
  -72.4876  186.402  .000000
  -107.460  168.678  .000000
  -138.303  144.473  .000000
  -163.830  114.715  .000000
  ```

  will show up like in Fig.38(left).

- If such segments are separated by 1 blank line, like,

```
-34.7296  196.962  .000000
-72.4876  186.402  .000000
-107.460  168.678  .000000
-138.303  144.473  .000000
-163.830  114.715  .000000

-34.7296  196.962  500.000
-72.4876  186.402  500.000
-107.460  168.678  500.000
-138.303  144.473  500.000
-163.830  114.715  500.000
```

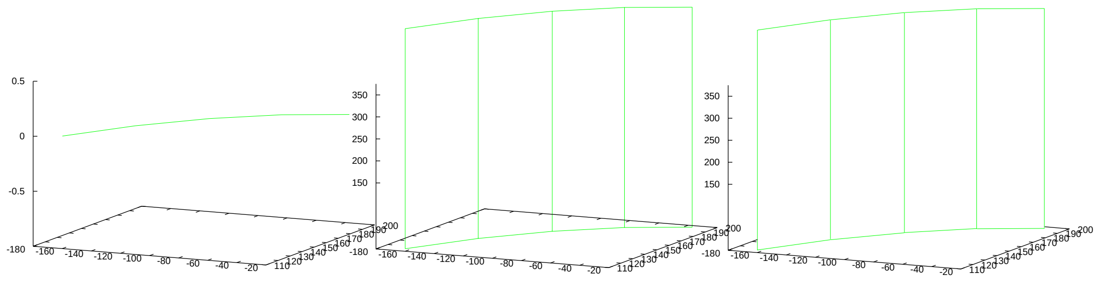


Figure 38: Left: Consecutive (x,y,z)'s draw a line. Middle: If two of such a group are give with one blank line inbetween, the nodal points are connected. Right: For such a display, hidden mode is possible.

then, corresponding points in each group are connected by a line at display time (If the number of points in each group differs, error takes place). (Fig.38(middle)). At this stage, you can specify the 'hidden mode' (Fig.38(right)).

- If two or more blank lines are inserted in the data, those in the separated groups are mutually not related.
- In your coding, you have to give numerical values in 'p' (not a text like the example shown above). **For a blank line, you have to give a special value which is contained in 'gpsep' defined in the system**. For example, p(10).x = gpsep. No value may be given to the y and z components in this case.
- As n, you have to return the number of rows including the blank lines.
- Several predefined routines are available for use:

  epdrawCcl; to draw a circle.

  epdrawCylEdg: to draw a cylinder cross-section.

  epdrawPipeEdg: to draw a pipe cross-section.

  epdrawSphere: to draw a sphere.

  epdrawElps: to draw a ellipse.

  These have parameters that permit to draw a part of the volume.

After completing the program, you may iput it in *Epics*/Util/DrawNewVol/

14.2.3 How to test your coding

- After you make ep_xxxx.f, you have to incorporate it in the library.

  - As describe earlier, you may replace, e.g, epdummy1.f in *Epics*/prog/NewVol/ by your program file.
  - Then, in that directory, you may change all 'epdummy1' in Makefile to 'ep_xxxx'.
  - After that, 'make' in that directory will work for updating the library.

- After successful compilation of your program, you may go to *Epics*/Util to test your coding for particle tracking.
- Issue,

  `make -f testCnf2.mk`

  to make a.out.
- Invoking a.out will ask you options.
- The program can do two different tests:

  One is **boundary test**. The system generates a number of random points inside the world, draws random half lines[15] from such points and see if the boundary can be obtained or not. If the boundary is found, the system write the boundary coordinate (x, y, z). (If DEBUG option is alive, two more integers are added). If you display such points, it should show the volume-shape you designed.

  The other is **in-out test**. The system generates a number of random points inside the world and test whether they are inside or outside the given component. The system output is (x,y,z), icon, cn, where, icon is 0 if the point is inside else 1, cn is the component number. You can verify that your coding is valid or not by displaying the points and examining icon.
- You should remember that the tests shown above are not perfect, though you will be able to eliminate most of the bugs.
- The coding for volume shape drawing can be tested by making 'drawconfig'.

15 Inquiry routines

When you make your own application, you sometimes need information not known from the parameters given by userhook. In that case, you may use some inquiry subroutines. They are listed at the top part of ephook.f

16 Parameters for user's use

There are three sets of parameters which the user can use freely for any purpose. They may be placed either at the last part of 'epicsfile' or 'sepicsifle' ; unlike other parameters, their entry may not exist if they are not used. **If some entries are given in 'sepicisfile', all of the entries, if any, in 'epicsifle' are ignored.**

```
 epHooks  '2  1  3'
 epHookc 'afile'
 epHookc 'bdirectory/bfile'
 epHooki   -1
 epHookr  10.
 epHookr  3.14d5
 epHookr  5.0
```

[15] To draw a random half line, you can choose the isotropic angle or angle that is exactly parallel to one of the (x, y, z) axes.

The first epHooks ('2 1 3') shows the number of epHookc, epHooki, and epHookr lines, respectively. Each of epHookc entries (2 in the above example) gives a maximum of 100 character data enclosed by quotation marks. Next epHooki is for ineteger data. epHookr is for real*8 data. A maximum of 5 , 10, and 10 entries in default can be given for epHookc, epHooki, and epHookr, respectively. If epHooks were '2 0 3', there would be no entry of epHooki.

The user can access to the value of the i-th entry of each category by the following subroutine calls:

For epHookc : call epqHookc(i, cv) where cv is a character variable with the length containable the given data.

For epHooki : call epqHooki(i, iv) where iv is an integer variable to accept the given data.

For epHookr : call epqHookr(i, rv) where rv is a real*8 variable to receive the given data.

If 'i' is out of range, cv will be blank, iv -9999999 and rv $-1.d-60$. The user can change the default size of the number of entries by modifying `EPMAX_UHOOKC, EPMAX_UHOOKI, EPMAX_UHOOKR` in *Epics*/epics/ZepMaxdef.h.

17 Drawing the basic cross-section formulas and testing random sampling for the processes

Caution: **DEC Fortran users** may have some trouble with the output in this section; a long line is automatically put into 2 lines and subsequent processes such as drawing graphs will fail. In that case, you have to specify the format in the write statement.

Go to *Epics*/Util/Elemag. To make the 'echoErr' command, do the below once for all. This is the same one implicitly used when distributing Epics or Cosmos jobs over a number of workstations.

```
cc -o echoErr echoErr.c
```

17.1 Drawing the formulas

Go to *Epics*/Util/Elemag

- To draw bremsstrahlung cross-sections, use

  DrawBremsFunc

  script. The output for one line is

  $xE_e/(E_e - m_e), \quad f \times x, \quad f, \quad E_e$

  where E_e is the electron energy, m_e the electron mass, x the fractional energy of emitted photon (E_γ/E_e), f the cross-section value (for unit, you have several options). If you specify more than 1 energy, the data is separated by 1 blank line.

- To draw pair creation cross-sections, use

  DrawPairFunc

  script. The output for one line is

  $(xE_\gamma - m_e)/(E_\gamma - 2m_e), \quad f, \quad E_\gamma$

  where x is $E_e/E_\gamma(\geq 0.5)$. The first field will take a value from 0.5 to 1.0.

17.2 Testing random sampling

Go to *Epics*/Util/Elemag.

- To test random sampling of bremsstrahlung,

  make -f BrSamp.mk

  Then, a.out is created. The output from a.out for each line is

  $$E_\gamma/(E_e - m_e), \quad \log_{10}(E_\gamma/(E_e - m_e)), \quad \text{path}, \quad \text{prob}$$

  where E_γ is the sampled photon energy, E_e the input electron energy, 'path' the sampled random path length (in radiation length), and 'prob' is the probability of bremsstrahlung occurrence per radiation length.

- To test random sampling of pair creation,

  make -f PrSamp.mk

  Then, a.out is created. The output from a.out for each line is

  $$E_e/E_\gamma, \quad (E_e - m_e)/(E_\gamma - 2m_e), \quad \text{path, prob}$$

  where E_e is the higher electron energy, 'path' the sampled path length in radiation length, and 'prob' is the probability of occurrence of pair creation per radiation length.

- To test Compton scattering,

  make -f Compton.mk

  An executable 'a.out' generated by the command above will write on each line:

  $$\text{prob}, \quad E'_\gamma \quad E_e \quad \cos_\gamma \quad \cos_e \quad \text{path}$$

  where 'prob' is the probability of occurrence of Compton scattering per radiation length, E'_γ the scattered photon energy, E_e the scattered electron energy, $\cos_\gamma$ the cosine of scattered photon angle, $\cos_e$ the cosine of scattered electron angle, 'path' the sampled path length in r.l.

- To test dE/dx of Urban model.

  make -f urban.mk

  a.out will put

  $$\Delta E, \quad \Delta E/{<}\Delta E{>}/\text{thickness} \quad \Delta E/\text{thickness}$$

  where 'thickness' is dx in cm and energy is in MeV.

- Others. You can see various test routines in *Epics*/prog/Elemag/UC. (say, for Bhabha and Moller scattering, knock-on process, positron annihilation, photoelectric effect).

18 Miscellaneous

18.1 Ready-made media list

The following materials are available for use. If you cannot find your material here, you can make it easily following the prescription in Sec.13 (parentheses are comment and not a part of the name):

Acrylic Air Al Argas BGO Bakelite CH4 Carbon (old one was C) **CsI Cu Fe G5** (nuclear emulsion) **Gten** (old one was G10) **H2 H2O Hegas LiqH2 LiqXe Ni Pb SCIN** (plastic scintillator) **SciFi** (the same as SCIN) **Si StdRock** (standard rock) **Ta W hollow** (the same as sp) **sp** (almost vacuum H2 gas) **YBAir**(the same as Air but the density is at YangBajin, China). You may employ 'hollow' or 'sp' to distingush, say, the world and some space components.

18.2 Config file format

#eq Example:

#eq cos60 0.5

Equates a symbol to a numerical value (mathematical expressions are not permitted). The symbol must start with a alpha character. A maximum of 16 alpha-numeric characters can be used for a symbol. A maximum of 32 symbols can be defined.

#subd Example:

```
#subd   abc
1  box   Pb
2  cyl     Al
       ...
#end  abc
```

Subdetector name must be a maximum of 8 lower case characters. It must not coincide with any of the first part of the existing volume-shape names; i.e., box, boxwide, boxskew etc are not allowed while widebox, skewbox etc are allowed.

In the definition of a subdetector, we can include other already defined subdetectors. When a subdetector is included, the material part may be 'sp'. There is no attribute field. The direction cosines can be given as if the subdetector is a box.

A maximum of 25 subdetectors can be defined.

#news Example:

#news new-5 skewbox

Declare to use a new volume-shape defined by the user. 'cone', 'ecyl' and 'cap' are regarded as the user defined volume-shape and you need #news to use them. The naming rule is the same as the #subd. For a really new volume-shape other than 'cone', 'ecyl' , 'cap' and those listed in *Epics*/prog/NewVol/, you need to write a handler for them (see Sec.14.1). A maximum of 15 new volume-shapes can be used.

#com Example.

#com 0

will suppress comment output from 'drawconfig'.

#com 1

will activate comment output again. The default setting is 1. Comments indicted by /* */ are not subject to this rule.

#do Example.

```
#do 6
25  withpb  sp 0 0 0 / 0 0 +
c    fiber xy 3
26 xybelt sp 0 0 0 / 0 0 +
#end do
27  box  ...
```

Repeat 6 times those inbetween #do an #end do. Note the nubmers you give at the line top is apparent one. This must be pre-processed, for example, as

expando configWithDo > configWithoutDo

to obtain a normal configuration file 'configWithoutDo' . An example is in UserHook/BetsDetail2/config.

renum . In some case, we have to renubmer many of cofing file lines by some reason or other. This is a bothersome task. 'renum configfile' will renumber the data lines. (The input file to renum must have at least some number in each data line).

The general format of one line describing a component in a config file is as follows:

type matter c de maxpath ch# / x y z [a b c] [dir's] [/ matryoshka_list]

Each field is described below where ['=', 'i' etc] means you can use one of such notations. ('s' means a symbol #eq-uated to an integer or floating value appropriately. 'i' means an integer value, 'f' floating value. For a positive number, notations like +2.45 cannot be used; simply use 2.45).

Sequential number. [i].

Must be consecutive from 1.

type Predefined volume-shape name, #news-defined new volume-shape name, or sub-detector name. A maximum of 8 lower case characters.

matter Material name. A maximum of 8 characters that can be a file name. When referring to a subdetector, 'sp' may be used.

c Give one of 0, 1, 2, or 3. [i, =,s].

'=' means the value of the previous component is used.

0 means you don't want to count particles entering into/exiting from this component.

1 means you want to count particles entering into this component.

2 means you want to count particles exiting from this component.

3 means you want to count particles both entering into and exiting from this component.

For non zero value, 'userbd' is called.

de Give one of 0,1, 2, -1, -2. [i, =, s]. 0 means you don't want to count energy loss in this component. (Internally, average energy loss is computed).

1 means you want to count energy loss in this component. You don't need to consider the energy loss fluctuation.

2 means you want to count energy loss in this component. You want energy loss fluctuation.

-1: the same as 0.

-2: the same as 2 but userde is not called.

maxpath Give a maximum movable distance of a particle in this component. Give some value if you want to limit the path length by some reason or other. [' ', f, i, =, s]. If 0 or ' ' is given, system takes care of it automatically. (' ' means blank.) For a very thin foil such as few 10's μm or less, try 1/2 $\sim$1/5 of the foil thickness and see the results.

ch# Optional channel number of the component. For a given component, 'comp', comp.ch contains the channel number. If it is a component without having channel number, -9999999 will be put.

/ Before x, this is mandatory. It may be placed just after a number, like 235.6/ but not come just after a #eq-uated symbol like maxp/

x, y, z Give the origin of the component in the world coordinate. [i, f, s, =, +, =i, =f, =s, +f, +i, +s, ~f, ~i, ~s].

'=' means the value is the same as the previous component, '+' previous origin + thickness of the previous component (for x column, the x-thickness), '~' previous origin − thickness of the current component.

'=i' means offset 'i' is added to the case of '='. '+i' means offset 'i' is added to the case of '+'. '=f', '=s' etc are similarly defined.

a b c Give attributes which determines the given volume-shape. [f, i, =, s]

The number of the attributes depends on the volume-shape. For a sphere, it is only one (radius). There is no attributes for a subdetector.

dir's For a non canonical volume-shape, give direction cosines to show the orientation of the volume. [f, i, s, =]

Their number will be 0, 3 or 6. For a canonical volume or derived canonical volume (such as cyl_x, prism_xz etc) , they need not be given. If the first three are not (1, 0, 0) and second three are (0, 1, 0), you may only give the first three.

/ matryoshka list If the component (partially) contains other components, give the list of components numbers ([i]) after putting '/'. The '/' may be attached to a preceding number like

'1.45 2.45/'

but must not be like

'1.45 cos25/'.

(You have at least one space like '1.45 cos25 /').

If the component partially contains other components (sort of clipping), negative sign must be added to those component numbers. (like / $-25-10$).

In the case of the explicit world (box_w or sphere_w), the origin may be 0 0 0. You need not give attributes. If you give both origin and attributes for the explicit world, they may be employed only if their size is larger than the default minimum size.

18.3 Getting expanded config data

When you select menu 8 of 'drawconfig' in *Epics*/Util, you will get the detector configuration in the final expanded form. In some case (for example, to make correspondence between the component number and channel number), you might want to re-direct the output. For this end, you may use

make -f testCnf1.mk
echo *config* | ./a.out

18.4 Stacking a large number of particles as incident

If you want to make a group of particles as incident particles, you may make a track information of each particle (=aTrack) and call epputTrack(aTrack) as many times as you want. However, if the number is too large (say, > 3950), the stack area might overflow. You could incease the limit by changing the maximum value in *Epics*/epics/ZepMaxdef.h.

The other more convenient method is to call epputTrk2(io, aTrack) as many times as you want. Here, 'io' is an integer indicating a logical fortran device number. It must be linked to a binary file and the file must be opened by the user. Typically, the user may open it in uiaev, rewind it in ui1ev and put tracks in usetip. When all the particles are put, the user must 'rewind io'.

18.5 Particle code list

Epics uses the same particle code as Cosmos does. A particle is identified by the particle code, subcode and charge. When you need to identify a particle in the user hook routines, you may use the `#include "Zcode.h"` directive and refer the code names in that file rather than code numbers.

The following list is the names that represent the particles in Cosmos. They are roughly in the order of mass.

particle	code name	code number	particle	code name	code number
photon	kphoton	1	electron	kelec	2

muon	kmuon	3	pion	kpion	4
kaon	kkaon	5	nucleon	knuc	6
ν_e	kneue	7	ν_μ	kneumu	8
deuteron	kdeut	9	triton	ktriton	17
He	kalfa	10	LiBeB(A∼8)	klibe	11
CNO(A∼14)	kcno	12	H(A∼25)	khvy	13
VH(A∼35)	kvhvy	14	Fe(A∼56)	kiron	15
D meson	kdmes	16	ρ	krho	24
Λ	klambda	18	Λ_c	klambdac	21
Σ	ksigma	19	Ξ	kgzai	20
ω	komega	25	ϕ	kphi	26

The sub-code is used to discriminate the particle from anti-particle, if the difference is essential such as the neutron and neutrino, but not used for, say, anti-protons because the charge can tell it. For K0 mesons, the sub-code is used to distinguish between K0-short and K0-long. Cosmos does not assign the particle and anti-particle code to them. They are assumed to be produced in equal weight and the actual assignment is performed randomly when they interact. To identify the particle and anti-particle, you can use the sub-code name, "regptcl" and "antip". For K0-short and K0-long, the sub-code name "k0s" and "k0l" may be used.

18.6 The KF code and Cosmos code conversion

The Cosmos particle code can be converted into the KF code recommended in Particle Data book and vice versa. To convert a KF code, 'kf', into Cosmos code (code, subcode, charge)

```
call ckf2cos(kf, code, subcode, charge)
```

where all variables are integers. The inverse is

```
call ccos2kf(code, subcode, charge, kf)
```

18.7 Extending Epics default

In some case, you may need much more detector components than default maximum number. In such a case, you can extend array size etc. The definition of default maximum values of various quantities is found in *Epics*/Epics/ZepMaxdef.h. Comment there will tell you which one should be extended. After changing it, re-make the library.

A Major differences from the older versions: as of uv 5.04

Some of the new features of the current Epics is that

1. You can use a variety of (volume-)shapes as detector components. If the pre-defined ones do not satisfy your requirement, you can make a new shape by combining existing shapes or by using a sort of clipping. If these still do not match your needs, you can define completely new shapes (some coding is required in this case).

2. Basic electro-magnetic cross-sections are treated more accurately than the older versions. If a material you need is not defined in Epics, you can easily define it and add it to the Epics media list. There is practically no upper limit in the energy, though no body knows, for example, the LPM effect at as high as 1000 TeV or more in lead is valid as it is.

3. In defining a complex detector, convenient methods are available for use.

4. You can see a 3-D view of the detector to verify the configuration with various modes optionally together with simulated particle tracks.

5. You can see various basic cross-sections and simulated results for the processes graphically (good for pedagogical use).

6. If a number of homogeneous unix workstations is available for use, you can dispatch jobs over a number of such workstations and collect the results easily. The scripts for that purpose are formally supported in this version.