Summary of the Recent Updates of EPICS

K.Kasahara

September 22, 2011

Contents

This manual describes the recent update (for version 9.10) as well as some summary including old stuff and some obsolete parts of EPICS.

1 Remarks

v9.081 vs v9.10 There are rather many differences between EPICS version 9.08 and 9.10: If we should find some defect in v9.10, we might be forced to go back 9.08. However, some modification of v9.08 is mandatory. V9.081 is the minimum update of v9.08.

Emin and tracking scheme In V9.10, we introduced automatic Emin setting depending on the size of the detector component, special treatment of such setting (Emin and/or quenching factor) for a specified component, etc. To stop tracking of a particle, V9.08 used only Emin information but V9.10 could use residual range information. Results by V9.10 have been compared with those by V9.08 to find that there is no statistically significant difference between them (as to the energy deposit, $< 0.5\%$). If we use residual range information, the execution seed is improved by about 20∼30%.

New volume-shape As to the treatment of a detector component such as described by "pipe_y", there is some subtle difference between 9.08 or earlier and 9.10. V9.10 introduced new shapes (octagon and honeycomb which permit "_y" type notation)[1]. This cannot be treated by V9.08 without some modification of the source code. (The maximum character length of the shape name was changed from 8 to 12). V9.081 is a version that can understand this new feature with a minimum change of V9.08.

Cosmos version The following combination must be used. Cosmos7.581 is little bit

Table 1: default

EPICS	Cosmos
9.081	7.581
9.10	7.60

different from 7.58; mis-conversion of η and Λ^0 code from QGSJET-II was corrected. Completely stopped anti-proton cannot annihilate in the Jam code and this leads to infinite loop. Although very rare, π^0 makes a collision in Jam which cannot treat it. These were corrected. Cosmos7.60 could use the "sofia" code for photo-hadron production.

Jam code The Jam code in 9.10/9.081 is the same as in older EPICS's (Y. Nara, Nucl. Phys. A 638, 555c (1998));

It has some problem with the treatment of spectator nucleons in heavy ion collisions. Spectator nucleons in a projectile or target, emerge as independent nucleons after collision (even for elastic collisions). That means, for example, when Fe is a projectile, we will never get He secondaries after a collision. The current EPICS

[1]These new components and new treatment of "_y" type specification are described in another manual.

treatment of these nucleons is that we accept all nucleons from projectile while discard all spectator nucleons from target. (Let's call this Jam Jam1)

There is another Jam code embedded in the PHITS code[2] In this Jam, the spectator problem has been solved by the PHITS author . Let's call this Jam Jam2. However, Jam2 inherits some defect existed in the original Jam, say, K0 cannot be a projectile particle and other minor bugs. Such problems have been corrected in Jam1. **The implementation of Jam2 is under investigation**.

Jam1 and PHITS combination In the current versions, combination of
IntModel=`'"phits 2.5 "jam" 5 "dpmjet3"'`
was expected to improve the proton primary case at low energies (< 200GeV). However, the current tendency is rather opposite and contradicts earlier observation. This is under investigation.

Intel compiler vs VAX extension This is very much annoying stuff. Cosmos/EPICS are old and use structure construct based on the so called VAX extension[3]. The compiler seems to have a bug in dealing with the VAX style structure and in some case we encounter quit strange phenomena. Suppose a code fragment like

```
 structure /epPos/
      real(8):: x, y, z
 end structure

 record /epPos/  p(100)
 integer n
 real(8):: d
...
...
 p(n).x = 0.
 p(n).y = d
 p(n).z = 0.
```

In some case, even if "d" is non zero, p(n).y becomes 0. This dose not happening always, but seems to depend on other environment, So far we could not detect the condition for such happening.

One workaround is to write

```
 p(n) = epPos(0.d0, d, 0.d0)
```

`p(n).y = d` like coding is everywhere; in the case of detector drawing, it is easy to find the happening. However, in other pars, it is rather difficult to find the same. The users are recommended to use epPos(..) style coding in their UserHook.

[2]PHITS is a one complete package (K. Niita et al., Radiation Measurements 41, 1080 (2006)) for particle transport (at low energies). PHITS includes several interaction models including Jam. The interaction model specified by IntModel="'phits"' implies such models but does not include Jam. An appropriate model inside PHITS is selected depending on the energy and projectile type.

[3]Before Fortran90, there was no structure construct formally in Fortran. However, the C-language style structure has been used long time and it is called VAX extension. This extension is supported by the Intel Fortran compiler but Intel seems not serious about its support and the recent compiler says it will become obsolete in the future versions..

2 dE/dx of heavy ions

The ionization energy loss rate ($-dE/dx$; hereafter we regards dE/dx has a positive value) of heavy ions at low energies has been treated by an effective charge method and is fairly accurate down to a few hundred MeV/n where accelerator test experiments are usually performed. However, at lower energies, we need a more accurate treatment. We introduced two things:

- A better effective charge method (Pierce and Blann. Phys. Rev. 1968 vol.173,No2. pp.390-404. With later errortum). Some modification has been done for the He case.
- Incorporation of the SRIM data (`http://www.srim.org/#SRIM`). At present the data for plastic scintillator and SciFi are available. (They are currently regarded as the same media).

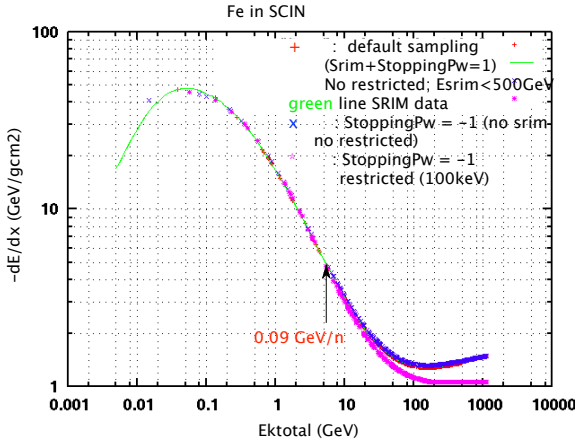

Figure 1: dE/dx of Fe in SCIN. Green line is by SRIM. Lower dots are the restricted energy loss rate with RecoilKeMin=100 keV. Upper one the full dE/dx. They coincide below 0.09 GeV/n. Blue crosses (x) by StoppingPw=-1 (new effective charge method) agree with the green line fairly well.

Table 2: Related parameters. D=xx means the default

variable	in	value	description
StoppingPw	epicsfile	D=1	When SRIM data is available at low energies (see `SrimEmax` below), use it. At higher energies, the effective charge method shown above is used.
		2	Same as above, but the old effective charge method is used.
		-1 or -2	Even if SRIM data exists, it is not used. The effective charge method corresponding to 1 or 2 is used.
SrimEmax	epicsfile	D=0.09	(GeV/n). Above this energy, SRIM data is not used. In the actual simulation, this value should not be much larger than this since the SRIM data is for the average of the total energy loss including high energy δ-rays from the knock-on process. We use the restricted energy loss and δ-rays above `RecoilKeMin` are randomly generated.
MAXHEAVYCHG	ZepMaxdef.h	30	In Epics/epics. Maximum charge that can be treated by SRIM. If changed, recompiling of all sources may be needed.
`MAX_SRIMMEDIA`	ZepMaxdef.h	3	In Epics/epics. Maximum number of media for which SRIM data can be used.

2.1 Creating a new SRIM data

If the user want to add more SRIM data to the existing data or to create new SRIM data for a particular media, Epics/Util/SRIM may be consulted. The Readme there will tell how to do. To see continuation of the SRIM data to the larger energy region, testdEdx.sh in Epics/Util/Elemag/dEdx may be used.

3 Minimum Energy

Before v9.10, the minimum energy of particles (gamma, electrons...) during the particle tracking is fixed by parameters given in `epicsfile`. Although it can be dependent on the particle type, it is unique and independent of media thickness. In some case we have a thick PWO and thin Si, or have to transport particles in a very long beam pipe before they reach the detector. In such cases, the minimum energy may be better to be dependent on the detector component.

In v9.10 or later,

- Automatic determination of the minimum energy is possible. The value is fixed for each component by considering its media and thickness.

- When the particle energy becomes lower than the minimum, the particle tracking is not necessarily stopped; in some case further tacking is continued as described in Table below.

- If the value fixed by the automatic way or old way is not satisfactory, there is a mean to fix the minimum for each particular component. See the `Modifier` section.

- The procedure for the automatic determination could be changed by the user. (Probably, such needs will be rare).

Basically, the default `Eabsorb` specifies that the kinetic energy of a charged particle and photon is absorbed at the point where the particle energy becomes lower than the predefined minimum. However, if the particle can decay or annihilate, EPICS may follow the particle down to 0 or some lower energy.

3.1 Automatic Emin

The automatic minimum energy is calculated in the subroutine located in

Epics/prog/UserMayChange/epAutoEmin.f

If `AutoEmin` is non 0, this program is called to fix the minimum for a given component. The default value, 2, specifies the following procedure.

- Use the input "minimum" thickness of the component in g/cm^2($= t$).

- Compute, max(min($150\sqrt{t}$, 10), 150)$\times 10^{-6}$ (GeV). That is, the value is always between 10 and 150 keV. The value is used for photons (EminG). For electrons, 2 times of this is used and electron mass is added (EminE). Later, the range consideration is applied.

- For AutoEmin=1, we compute max(min($100\sqrt{t}$, 10), 100)$\times 10^{-6}$ (GeV) and assigned to EminG and EminE.

- In both of the above cases, the value of `RecoilKeMin` is fixed by

  max(EminG, $14Z^2 10^{-9}$)

  where Z is the effective atomic number of the media. That is, the rough K-shell energy is considered (GeV).

- The values for `KEmin` is fixed to be the same as `EminG`

- `EminH` is unchanged.

If the user give a value > 2 to `AutoEmin` and add a program fragment in epAutoEmin.f, different treatments can be used. If it is 4, the range consideration will be done same as it is 2.

Table 3: Related parameters

variable	in	value	description
AutoEmin	epicsfile	D=2	The minimum energy is fixed automatically (see section for `Automatic Emin`). If the particle energy becomes lower than the minimum, the residual range of the particle is computed and compared with the distance to the boundary of the present component. If the range is smaller than the distance to the boundary, the energy is assumed to be absorbed within the range.
		0	The old method is used. The values (EminElec, EminGamma, KEmin, EminH) are listed below.
		1	Some smaller values of Emin than the AutoEmin=2 case are employed but we don't consider the range and treated as AutoEmin=0 case.
		3,4	Reserved for the user. For 4, the range is considered like AutoEmin=2 case.
EminElec	epicsfile	D=511e-6	(GeV) By historical reason, the minimum for electron is always in the total energy.
EminGamma	epicsfile	D=100e-6	(GeV) Photon minimum energy (different from the one for light)
KEmin	epicsfile	D=0	If 0, the minimum kinetic energy for electron is used. This is for the minimum kinetic energy for other particles than electron and neutron
EminH	epicsfile	D=0	If 0, 20 MeV is used for neutron minimum kinetic energy
RecoilKeMin	epicsfle	D=0	If 0, `EminGamma` is used. The restricted energy loss is computed below this energy and δ-rays are randomly generated above this energy.
Eabsorb	epicsfile	D=14	The bit pattern of `Eabsorb` determines how to treat energy of a particle when its energy becomes lower than the predefined minimum. For details, see `epicsfile` in UserHook/Template. See also below.

4 Modifier

The user may need to specify some specific minimum energy or non default quenching effect coefficients for some components[4]. In such cases, the user may give a number in the `modifier` digit for that component in the `config` file (see Fig.). The user must prepare a `ModifyFile` in which the user give that number followed by necessary entries

[4]The quenching coefficients are normally given in the media file and used as the default. If a modifier described here is to specify a change of the coefficients, the quenching treatment is applied even if there is no default specification.

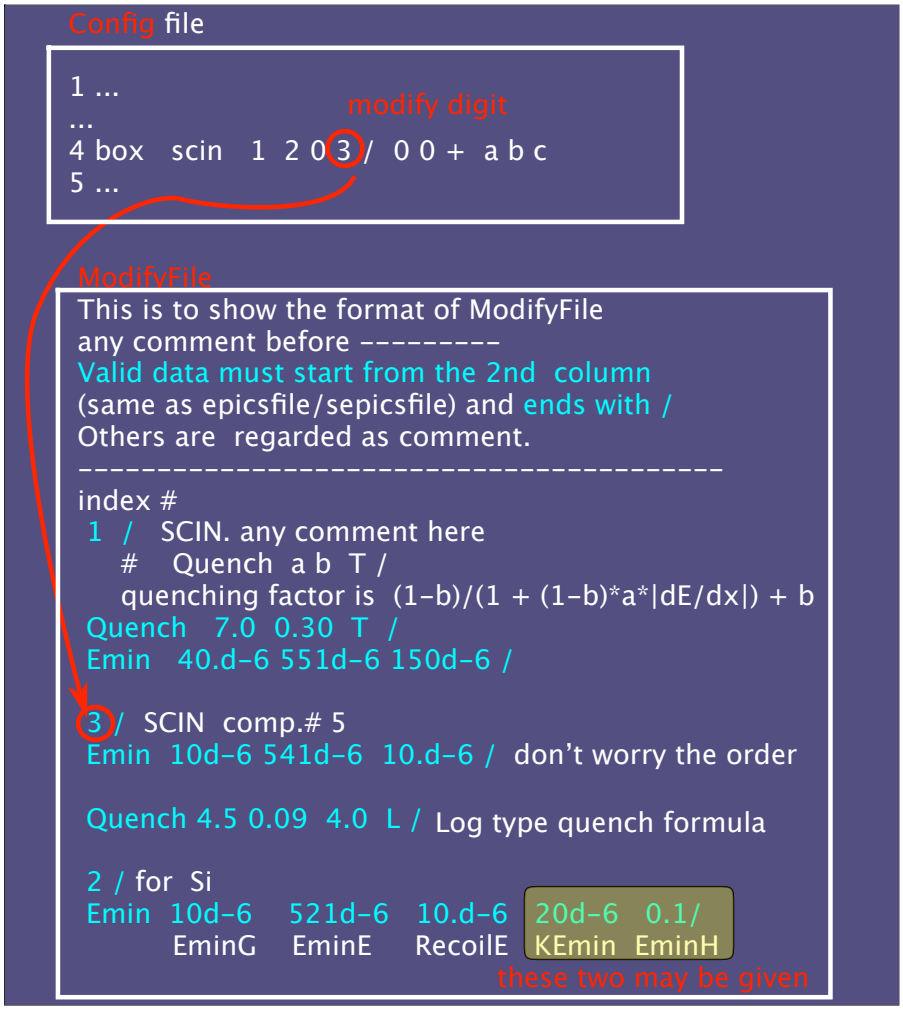

Figure 2: Modifier and ModifyFile

like in Fig. Note that, although we say as if `ModifyFile` were a file, `ModifyFile` itself is not the file name but a variable to contain a path to a file in which modifications are described,
If the modifier field is absent or 0, no modifier is assumed. The modifier number need not be consecutive but it's better to keep it as small as possible to save the memory (must be $< 10^{15} - 1$). The value for the `ModifyFile` must be given in `epicsfile`. The default of `ModifyFile` is ' ' so that no modifier is assumed. The format of `ModifyFile` is similar to `epicsfile`.

The current possible entry is `Quench` and `Emin`. In some case, one may need to change the medium density for the same medium (say, for Air), so `Density` could be an entry candidate. By historical reason, changing density is possible by the notation:

```
2 bos  Air  ...
3 box  Air*1.08  ..
4 box  Air*0.90
```

in the `config` file. Here, 1.08 means 1.08 times higher density than the default given in the media file. It should be noted, because of the LPM effect, changing the density dynamically is not an easy task so that this method must be used only if the LPM effect works very weakly.

Table 4: Related parameters

variable	in	value	description
ModifyFile	epicsfile	D=' '	If the modifier digit is used in the `config` file, a file name here is consulted. However, If this is "blank", all modifiers are neglected. The file should contain the number given in the modifier field and some of the variables shown below. If a modifier number is > the max number in `ModifyFile`, error stop will happen.
Quench	ModifyFile		Coefficients must be given for Tarlè, Birks or Log forrmula (see section for Quenching) . If this is missing for a modifier number, the same action is taken as if the modifier were absent.
Emin	ModifyFile		`EminG, EminE, RecoilE` must be given. `KEmin` and `EminH` may or may not follow them. If last two are not given, the same procedure as `AutoEmin` = 1 case is used. If this is missing for a modifier number, the same action is taken as if the modifier were absent.

4.1 Quenching coefficients

It is assumed that the amount of scintillation light emitted by a heavy ion in a short distance, Δx, is not proportional to the energy loss (deposit), $\Delta E = \frac{dE}{dx}\Delta x$, in the scintillator but is proportional to $C_f \Delta E$ where $C_f (\leq 1)$ is a dE/dx dependent constant.

Birks Before v9.08, the quenching effect is managed by the Birks formula + some corrections. The original Birks formula gives

$$C_f = \frac{1}{1 + a\frac{dE}{dx}} \tag{1}$$

where a is the Birks coefficient. The additional corrections need two more constants, b and c. So, for example, the basic media file (Epics/Data/BaseM/SCIN) and media file (Epics/Data/Media/SCIN)[5] contains lines like

```
#  Elem  rho(g/cm^3)    Gas/Solid(1/0)  refl.index     Birks c
     2      1.032              0          1.581      13   9.6  0.5714
```

where the last three numbers are the coefficients, a, b, c. However, the correction terms using b and c do not work well and only a has been used in the original formula to get C_f. In spite of this fact we will keep 3 numbers for the Birks case; the last two may be any numbers.

Now we may put "B" to express explicitly that these are for the Birks formula:

[5]The `BaseM` file is used only when making the `Media` file. The user may change the quenching coefficients in the media file after creating it. The data in the `BaseM` file need not be changed but it will be better to keep the same value as the `Media` file.

```
#  Elem  rho(g/cm^3)    Gas/Solid(1/0)  refl.index     Birks c
     2      1.032             0           1.581   13  9.6  0.5714 B
```

The unit of a is g/cm^2/GeV.

Talré A better formula by Talré is now usable (G. Tarlé, S.P . Ahlen and B.G . Cartwright, Astrophys . J. 230 (1979) 607):

$$C_f = \frac{1-b}{1+(1-b)a\frac{dE}{dx}} + b \tag{2}$$

and the (basic) media file format is (e.g., for $a = 8$ and $b = 0.35$)

```
#  Elem  rho(g/cm^3)    Gas/Solid(1/0)  refl.index    Talre c
     2      1.032             0           1.581     8  0.35  T
```

We need two coefficients, a, b and "T". The unit of a is the same as the Birks case, i.e, g/cm^2/GeV and b is unitless.

Log Another purely empirical formula is a "Log" type[6] which needs three coefficients a, b and c:

$$z = a\frac{dE}{dx} + 1 \tag{3}$$

$$C_f = z^{-b\log(cz)} \tag{4}$$

The (basic) media file format would be

```
#  Elem  rho(g/cm^3)    Gas/Solid(1/0)  refl.index    Log quench
     2      1.032             0           1.581     4.6 0.09 5.1  L
```

The three coefficients, a, b, c, must be followed by "L". The unit of a is as before (g/cm^2/GeV) and b and c are unitless.

So far the format is for the (basic) media file and the values there are used as default for that medium. As mentioned earlier, the *modifier digit* and `ModifyFile` can change these defaults. The format in the `ModifyFile` is one of

```
Quench a b c B  /
Quench a b T /
Quench a b c L /
```

Q... must start from the 2nd column.

4.2 User own quenching treatment

If the user wants to use another quenching formula, one solution is to do every thing in `userde` of ephook.f. The coding will look like

[6]Coefficient treatment in v9.08 is different from v9.10 or later so the "Log" formula should not be used in v9.08.

```
real(8):: dedx, Cf
....
if( aTrack.p.charge > 1) then
  call epqElossRate(dedx) ! get dE/dx (GeV/(g/cm2)
   ! get Cf from dedx etc
   ! get effective dE by Cf*Move.dE
   !  use it instead of Move.dEeff
endif
```

Caution: If `info` given to `userde` is 1, the particle is dying, or already dead because its energy is < minimum from the birth. In the latter case, unless charge is > 1, `dedx` is undefined. In some case, even charge 0 particle (e.g, very low energy photons) may come there.

4.3 Getting the sum of energy deposit and effective energy deposit

The user can use (in `ue1ev of ephook.f`)
`call epqEloss(i, dEt, dEeff)`
to get true energy deposit (`real(4)::dEt`) and effective deposit (`real(4)::dEeff`) for a component number `i`.

Caution: dEeff may be taken to be proportional to emitted light intensity. However, actual light reaching to sensor could be dependent on the emission position. Such a factor is not taken into account in this dEeff. The user must do such business in `userde` using Move.dEeff and position information, etc

5 Seeing the minimum energies and quenching coefficients

To have a look at the minimum energies and quenching coefficients set by `ModifyFile` or `AutoEmin` together with the associated component, the user may go to Epics/Util and issue

- `./testCnf4.sh` for Emin
- `./testCnf5.sh` for quench coef.

The usage will be shown by the command.

6 New user interface

In some applications, the user may want to know information of particle interactions (what kind of interaction, where it happened etc). This type of interface is available in Cosmos but not in EPICS. If we add such one, all user must modify the existing ephook.f. This fact delayed implementation of such interface. Now, from v9.10, the user could use such interface[7] while those who don't need such one can use old programs without paying attention to, or without being aware of, the new interface at all.

How to do if the user wants to use the interface ?

1. Edit `Epics/epics/Zepcondc.h` and change the last line to read `#define INTINFO`

[7] If the user needs to know only the first interaction of the incident particle, this interface is not needed. See section 8.

2. `make clean;make` in Epics as usual[8]. This may be done once for all unless the user resets the line to `#undef INTINFO`.

3. In the user's application directory, say, `Epics/UserHook/myApps` or in `Epics/UserHook/xxx/myApps` (see 7. below), issue `manageIntInfo.sh`.

   This command dose two things

   - copies `Epics/UserHook/epUI.f` to the current directory.
   - after `#include "../main.f"` or `#include "../../main.f"` in the ephook.f, adds the next lines

     ```
     #include "Zepcondc.h"
     #if defined (INTINFO)
     #include "epUI.f"
     #endif
     ```

4. If the user dose not do anything more, and `make clean;make`, then the application will run as if the user did not do `manageIntInfo.sh`. The overhead by introducing epUI.f is completely negligible.

5. If the user wants to do something when some particle interacts, the user must edit epUI.f.

   - The usage is explained in the epUI.f
   - Important: Suppose a photon interacts and this program unit is called, then if the user dose not give 0 to `info`, this program will not be called for later photon interactions until the next event simulation starts. For other particles, the same is true.
   - The position information is the one at the local coordinate of the current component.

6. For the applications in `UserHook/` supplied by v9.09, `manageIntInfo.sh` has been done. If the user dose this again, nothing will happen.

7. If the user's application is located in a different directory structure than shown above, the equivalent two things as above must be done by the user.

The epGUI subroutine is included in the epUI.f. This is prepared to cope with future demand of new interfaces.

7 New volume shape

See a separate manual.

8 Inquiry and other useful subroutines

Here we list inquiry subroutines which may be needed by the user, irrespectively of old or new.

[8] This should be always ok, but if the user has already made the library before changing `Zepcondc.h`, the safest way is to delete the library in `Epics/lib/.../` once and remake the library.

Table 5: Inquiry subroutines

subroutine name	description
epqversion	Get version number of Cosmos and EPICS. Usage: `character(8)::cosv` ! Comos version (say 7.99) `character(8)::epiv` ! EPICS version `call epqversion(cosv, epiv)` Caution: Environmental variable COSMOSTOP and EPICSTOP must have relevant values.

epqncp	Get the total number of components.
	Usage:
	`integer::ncomp`
	`call epqncp(ncomp)`

epqevn	Get current event number
	Usage:
	`integer::eventn`
	`call epqevn(eventn)`
	The event number will be updated after `ue1ev` is called.

epqinc	Get the incident particle track information
	Usage:
	`#include "ZepTrack.h"`
	`record/eTrack/ aTrack`
	`call epqinc(aTrack)`
	aTrack.p.code etc are explained in userde or userbd. E.g, the component number of the incident track is aTrack.cn and its position is aTrack.pos (local coordinate). To get the world coordinate, the user must use `epl2w` (see Table 6)
	Caution: If multiple particles are incident, only the first one is obtained.

epqFirstI	Get the first interaction point of the incident particle
	Usage:
	`#include "ZepPos.h"`
	This is not needed if `ZepTrack.h` is used for `epqinc` in the same subroutine.
	`record/epPos/firstpos`
	`call epqFirstI(firstpos)`
	firstpos.x, firstpos.y, firstpos.z are the interaction point in the **world coordinate**.
	Caution: Except for the electron, the knock-on and elastic collision are not regarded as interaction. A large negative value (-1000000.0) of `firstpos` indicates no interaction happened.

subroutine name	description

epqFirstM	Get the media information in which the incident made the first interaction.
	Usage:
	`#include "Zmedia.h"`
	`record /epmedia/ firstM`
	`call epqFirstM(firstM)`
	Some of the ingredients:
	`firstM.A`: real(8) ! Average mass number of the media.
	`firstM.Z`: real(8) ! Average charge of the media.
	`firstM.name`: character(8) ! media name.
	`firstM.colElem`: integer
	The element # within the media at which the first collision took place. For e/γ/μ, this will be 0 if the number of elements in the media is > 1, and the **variables below are undefined**.
	`firstM.colA`: integer
	Mass number of the nucleus at which the collision took place.
	`firstM.colZ`: integer
	Charge number of the nucleus at which the collision took place.
	`firatM.colXs`: real(8)
	The inelastic cross-section (mb) of the target.

epqFirstP	Get process name of the first interaction.
	Usage:
	`character(8):: proc`
	`call epqFirstP(proc)`
	If no interaction happened, `proc` will be ' '. Hadronic collision is 'coll', brems 'brem', pair creation 'pair', Compton 'comp'; more details are seen in Epics/UserHook/epUI.f.

epqCn2Media	Get media information of a given component number.
	Usage:
	`#include "Zmedia.h"`
	`integer:: compn` ! give some number
	`record /epmedia/ mediax`
	`compn= ...`
	`call epqCn2Media(compn, mediax)`
	If `compn` is wrong, stop will happen. The media name is in `mediax.name` (see, `epqmat`), `mediax.Z` the average charge of the media, etc.

epqmat	Get media name of a given component number.
	Usage:
	`integer:: compn` ! give some number
	`character(8)::name` ! If `compn` is wrong, stop will happen.
	`compn= ...`
	`call epqmat(compn, mat)`
	Output `mat` will be such as `Pb, SCIN` etc.

subroutine name	description

epqstruc	Get component structure (box etc) of a given component number.
	Usage:
	`integer:: compn` ! give some number
	`character(12)::struc` ! If `compn` is wrong, stop will happen.
	`compn= ...`
	`call epqstruc(compn, struc)`
	Output `struc` will be such as `box, ciecone` etc. If the length is shorter than actual one, the tail part will be lost.

epqSubdName	Get sub-detector name to which a given component number belongs.
	Usage:
	`integer::compn` ! give some number
	`character(16)::name`
	If `compn` dose not belong to a sub-detector, name will be ' ' .
	`compn= ...`
	`call epqSubdName(compn, name)`

epqmatrhoc	Get media name (e.g, Air*1.03) of a given component number.
	Usage:
	`integer:: compn` ! give some number
	`character(20)::name`
	Output will be `Air*1.0032` etc. `Air` is equivalent to `Air*1`. If `compn` is wrong, stop will happen.
	`real(4)::rhoc` ! output.
	The relative density of the component to the default.
	`integer::lc` ! output. Length of the name content.
	`compn= ...`
	`call epqmatrhoc(compn, name, lc, rhoc)`

subroutine name	description

epqCount	Get digit information for calling `userbd` and `userde` of a given component number.
	Usage:
	`integer::compn` ! give a component number
	`integer::countio` ! digit for `userbd`
	`integer::countde` ! digit for `userde`
	`compn= ...`
	`call epqCount(compn, countio, countde)`
	Wrong `compn` will result in a stop. Suppose a component description line like
	3 box Pb c de / ....
	here "c" is the digit used to call userbd and "de" the one used to call userde (and routines for light transport). We call the "c" part **countIO** (since it is to specify whether `userbd` is to be called when a particle enters In or goes Out of a component), and the "de" part **countDE** (since it is to specify whether `userde` is to be called when a particle Deposits Energy in a component).

epLightUnpack CountDE	Decompose `countDE` (see above). Decomposition may be needed when it contains information for light generation and transport. `integer(2)::info` ! NOT integer `integer::d` ! digit for energy deposit count `integer::mn` ! digit for flle for Light or sensor `integer::B` ! digit for light generation/transport `info = countde` ! convert to 2 byte integer `call epLightUnpackCountDE(info, d, mn, B)` The input `info` is decomposed into `d, mn, B`. Details will be explained in the Light Transport section.

epqvolatr	Get the volume attribute of a given component number.
	Usage:
	`integer::compn` ! give a component number
	`integer::na` ! Output. number of attributes obtained.
	`real(8)::vol(x)` ! Output.
	x must be $\geq$ `na` which is dependent on the volume (say, for box 3, cyl 2, octagon 4 ...).
	`compn= ...`
	`call epqvolatr(n, na, vol)`
	Example: for a box, vol(1), vol(2), vo(3) will be `a,b,c` of the canonical form. If `compn` is wrong , stop will happen.

subroutine name	description

epqcmpdircos	Get the direction cosines of a given component number.
	Usage:
	`integer::compn` ! give a component number
	`real(8)::dir(9)` ! Output.
	`compn= ...`
	`call epqcmpdircos(compn, dir)`
	The direction cosines of the component, showing how the canonical form is rotated. Suppose a canonical box surrounding the component. The rotation is expressed by the direction cosines of rotated canonical box's `a, b, c`: `dir(1:3)` are for `a`, `dir(4:6)` for `b` and `dir(7:9)` for `c`. If the component is not rotated `dir(1:9) =(1,0,0, 0,1,0, 0,0,1)`. If `compn` is wrong , stop will happen.

epqOrig	Get the origin coordinate value of a given component number.
	Usage:
	`#include "ZepPos.h"`
	`record /epPos/ orgin`
	`integer::compn` ! give a component number
	`compn= ...`
	`call epqorg(compn, origin)`
	The value of `orign.x, orign.y, orign.z` is the origin of the component in the **world** coordinate. That is, the origin of the component in its canonical form is shifted by this amount. If `compn` is wrong , stop will hapen.

epqElossRate	Get dE/dx of the current particle Usage: `real(8)::dedx` ! GeV/(g/cm^2) `call epqElossRate(dedx)` To be used in userde. See caution in 4.2.

Table 6: Other subroutines

subroutine name	description

epl2w	Convert position in the local coordinate into the world coordinate
	Usage:
	`#include "ZepPos.h"`
	`integer::cn` ! input. component number
	`record /epPos/ posl` ! input.
	The position `posl` in the local coordinate of the component specified by the component number `cn`.
	`record /epPos/ posw` ! output. world coordinate position
	`call epl2w(cn, posl, posw)`

subroutine name	description

epw2l	Inverse of `epl2w`
	Usage:
	`#include "ZepPos.h"`
	`integer::cn` ! input. component number
	`record /epPos/ posw` ! input. world coordinate position
	`record /epPos/ posl` ! output. local coordinate position
	`call epw2l(cn, posw, posl)`

epl2wd	Convert 3 direction cosines in the local coordinate into the world coordinate
	Usage:
	`#include "ZepDirec.h"`
	`integer::cn` ! input. component number
	`record /epDirec/ dirl` ! input.
	Direction cosines `dirl` in the local coordinate of the component specified by the component number `cn`. `dirl=(dirl.x, dirl.y, dirl.z)`.
	`record /epDirec/ dirw` ! output. world coord. dir. cos.
	`call epl2wd(cn, dirl, dirw)`

epw2ld	Inverse of epl2wd.
	Usage:
	`#include "ZepDirec.h"`
	`integer::cn` ! input. component number
	`record /epDirec/ dirw` ! input.
	Direction cosines `dirw` in the world coordinate.
	`record /epDirec/ dirl` ! output. local coord. dir. cos.
	`call epw2ld(cn, dirw, dirl)`

subroutine name	description
cgetfname	Convert special characters in a string to create a new string. Usage: `character(x)::  fin` ! input `character(y)::  fout` ! output `fin='...'` `call cgetfname(fin, fout)` ! x,y must be some number. All of % #1 #2 # `@` $ in `fin` are treated as follows. 1) `#1` is replaced by the initial seed of the random number (1st one of the two). 2) `#2` is replaced by the initial seed of the random number (2nd one of the two). 3) `#` (not followed by 1 nor 2) is replaced by the unix process number. 4) `%` is replaced by YYMMDDHHMMSS (year month day hour minut second of the time). 5) `@` is replaced by the hostname (dropping domain name, if any). 6) `$` assumes it is followed by an environmental variable, and is replaced by its value. Three types can be recognizable: for instance, `$USER` (not followed by any character), `$USER/` (followed by /) `$(USER)` (always ok). In either case, it may be preceded by any character. (This one is usable from Cosmos7.59).
copenf	Open an existing sequential ascii file. Special characters in the file name is treated by cgetfname. Usage: `integer::ionum` ! input `integer::icon` ! output `call copenf(ionum, filepath, icon)` where `filepath` is a character string defined by, say, character(60)::filepath and contains a string showing the path to an existing file. `ionum` is the logical file unit number. `icon` =0: ok `iocn` !=0: could not be opened.
copenfw	Open an sequential ascii file for writing. It may not exist. Special character treatment is the same as copenf. Usage: `integer::ionum` ! input `integer::icon` ! output `call copenfw(ionum, filepath, icon)` ! icon =0: ok. else ng

subroutine name	description
rndc	Uniform random number in (0,1). Usage: `real(8)::u` ! output ($0 < u < 1.0$) `call rndc(u)` ! The same random number generator as used in Cosmos/EPICS. 0 and 1.0 are exclueded. There are two other generators and the third one is not used in Cosmos/EPICS. See for details in Cosmos/KKlib/rnd.f
Others	Other random number generators are available. See the following: (Those in Cosmos/KKlib) kgauss.f : Gaussina random number. kbetar.f : Random numbers with density of the beta function kbinom.f: Binomial random number. kcosn.f: cos and sin of uniform random number in $(0, 2\pi)$. knbino.f: Negative binomial random number. kpoisn.f Poisson random number. kampLin.f Random variable with density $(a + bx)dx$ ksampPEang.f: Random variable with density $(1-x^2)/(a-x)^4 dx$. Related to electron angle at photo-electric effect. ksampPw.f: Random variable from a function consisting of many power functions. ksampRSA.f: Random sampling of $\cos\theta$ from $(1 + \cos^2\theta)d\cos\theta$ ksbwig.f: Random sampling from the Breight-Wigner distribution. ksgamd.f: Sampling from the gamma distribution, $(x/a)^s \exp(-x/a)/\Gamma(s+1)d(x/a)$ ksplandau.f: Sampling from a psudo-Landu distribution: $\exp(-(y + \exp(-y))/2)dy$ where $y = (x - b)/c$. csampAF.f90: (in Cosmos/Module). Sampling from an arbitrary function specified by a numerical table. (see also ksampAF.f in Cosmos/KKlib/) **Those in Epics/prog/KKlib:** ksbeta.f: ksmpintbetaf: similar to kbetar.f ksx2.f: Random sampling from the χ^2 distribution.

9 Other updates and input parameters

- **Use of environmenal variables.** As described in Table 6, for the file name, we can use environmental variables.

- **Multiple scattering treatment.** The parameter `Molier` in the `epicsfile` now takes integer values rather than "t" or "f", although the user can still use t or f; they are mapped to 0 or 1 by

  f→ 0: This specifies the Gaussian multiple scattering.

  t→ 1: This specifies the Molière multiple scattering.

  The new possible value is 2. If 2 is given to `Molier`, Molière's multiple scattering formula is used, but it's implementation is completely different and more

rigorous than `Molier=1`. It also includes Bethe's prescription[9] to overcome the small angle approximation assumed in original Molière's theory. (Goudsmit and Saunderson's scattering formula[10] which does not use small angle approximation is well reflected in Bethe's prescription). However, results by `Molier=2` are (statistically) completely the same as `Molier=1` for cascade showers. The difference appears in the cpu time which is 1.6 times longer for `Molier=2` than `Molier=1`. Therefore, the user may use the default, `Molier=1`.

- **Automatic disk space allocation.** In older versions than 9.08, if the user inputs a large number of particles as the incident (using "+primary" file notation), the particle stack area could overflow during particle tracking; the user had to specify a disk file in the +primary file.

  In case of light tracking, the number of light photons becomes huge. Allocating more memory for the stack is not a good solution.

  Now, if the stack area lacks, EPICS automatically creates "scratch disk file" and the unix system deletes it at the end of job (even if the job ab-ends).

  The relevant parameters could be written in `sepicsfile`

  `StackDiskFile 'scratch' /`
  The default "scratch" does not mean the file name but it is a scratch file created by the system with some system-determined file name; the path to the file is fixed by the system;`ifort` will create it in `/tmp/$USER/` with the name something like fortVSXGZg. The file will be deleted at end of job (even if abnormal end).
  Caution:
  If a large number of jobs is submitted in a distributed system (e.g, pc clusters), depending on the system, many scratch files could be created in a non-local disk (say, in the NFS mounted home) to which network access is needed, then the jobs will almost kill the whole system due to overwhelming net work access. In such a case, the user must specify the path explicitly which does not require the network access. e.g, `/tmp/$USER/stackdisk#`. It should be noted, the user given file will be deleted only if the job ends normally, otherwise the user must delete it.
  For the scratch files, logical device number, 13 and 16, will be used in default.

- In Cosmos, some awkward behaviors in low energy interaction models were absorbed.

10 Interaction models

phits must be used with jam or nucrin or dpmjet3

11 Warnings

- Don't use InputP='fix' which requires Xinp etc. Instead, use InputP='u+z' etc with Xrange=... etc.

[9] Phys. Rev. Vol.15 (1953) 1256.
[10] Phys. Rev. Vol.57(1940)24, 58(1940)36.

- Don't put the incident exactly on the boundary of two consecutive components. EPICS will be buffaloed when judging the component the particle belongs to.

12 Light transportation