World and indexing by sub-detector hierarchy

June 26, 2014

1 Implicit vs explicit

- So far (< V9.164), world definition in a subdetector or the whole detector by using `_w` notation is permitted only for "box", "sphere" and "cyl" (`box_w, sphere_w` and `cyl_w` are the standard world).
- When the user wants to define a tight world in a subdetector and has to use non standard world such as "prism", "horse" etc (probably to avoid overlapping with other components), it must be defined at the last position of the subdetector as an *implicit* world.
- That is, without using the `_w` notation, the user has to explicitly list it's daughter component numbers after "/" (indirectly contained ones need not be listed). Besides, the user has to give its attributes, since it is difficult for the program to know how to place it (size, position and orientation).
- One problem of implicit world is that, when we show subdetector hierarchy by the *subdTree* command, implicit worlds are treated differently from explicit worlds, and the hierarchy structure looks somewhat obscure.
  The difference between explicit and implicit worlds are very small, but it is technically not so easy for the program to treat an implicit world as if it were an explicit (and hence the hierarchy structure looks the same as the explicit world case).
- As to the particle tracking, there is no difference at all between explicit and implicit worlds (expanded *config* files show the same structure).
- However, if we want to use hierarchy information to access particular components systematically (e.g, if we want to know the component number of the *I*-th SciFi in the *L*-th layer, etc by using the newly introduced *epGetIndex* subroutine), we need an explicit world for related subdetectors[1].
- From V9.164, any volume-shape (structure) can be the world of a subdetector (Note however, we need special caution when using "pipe", "honeycomb" etc which have "holes").
- Besides `cyl_w`, we can now use `cyl_x_w, cyl_y_w`. `cyl_z_w` is equivalent to `cyl_w`.
- The attributes (size, orientation etc) of the non-standard world must be explicitly written as in the case of the implicit world.
- Exceptions are the "cyl" case. If the attributes are not given, they are automatically assigned by referring to the bounding box information. If given, those by the user are used.
- Normally the daughter component numbers need not be listed; they are automatically searched for. However, if some daughter (A) contains another component (B), but the contained one (B) protrudes its mother (i.e, container) (A) (say, A is a pipe), we have to specify a container (C) of the protruded part (B') of B otherwise B' is not recognized properly.

[1] Explicit world is not mandatory. In some case, it is possible to apply *epGetIndex* even when target objects are contained in an implicit world.

If the world must play a role of such a container (C), we have to list the number for B explicitly, otherwise the program regards that B is already contained indirectly via A and omit the number of B. These are the same as in the past.

```
...
5  A  ...   / ...   / 6
6  B  ...  / ...
...
.  C_w ... / ...    / 6
```

- The name length of volume-shape (such as box, sphere etc) was 12. The current longest names are `honeycomb` (9) and `fpolygon` (8), so 12 was enough even for `honeycomb_yz` but `honeycomb_yz_w` exceeds 12. Therefore, this was now changed to 16 (same as subdetector name).

2 Alias of medium name

From the point of view of easy indexing of components in a detector, this is the same category as the indexing by using subdetector hierarchy (see Sec.3).

- The user can give aliases to a particular medium.
- For example, to SCIN an alias of CHD (CHarge Detector) may be given. Depending on the direction of CHD, the user may want to give CHDx and CHDy. Original SCIN is still usable.
- Alias must be specified in the config file before it is used like[2]

```
#alias   CHD SCIN
#alias   CHDx SCIN
#alias   CHDy SCIN
#alias   clad SciFi
1   box  CHD 0 2 0 / ...
2   box  CHDx 0 2 0 / ...
3   box  CHDy 0 2 0 / ...
4   box  SCIN 0 -2 0 / ...
...
5   box clad  0 -2 0 /   .../  6
6   box SciFi 0  2  0 / ...
```

- The alias name must be different from any of the existing media names and its length must be $\leq$ 8 (=`MAX_MEDIANEMELENG` in ZepMaxdef.h). One and the same name must not be assigned to different media names.
- The name is kept in "matter" part of the component. Suppose the component number is "cn", then Det.cmp(cn).matter keeps the alias of the medium of that component. If no alias is given, it becomes the true name.
- For a given component number, the user can obtain alias and true name of the medium of that component by

  `call  epqCn2AliasMediaName(cn, aname, tname)`

  where `aname` and `tname` are defined character variables with length ($\geq$ `MAX_MEDIANEMELENG`) to receive alias and true name.

[2]Here "clad" is fake; clad of SciFi must not be SciFi (refraction index must be < SciFi).

Note that

```
call  epqCn2MediaName(cn, tname)
```

is to get the true name. (One in the Media structure as Media(index).name is the tname).

3 Making an index table by using hierarchy information

Suppose a number of layers containing SciFi's in a detector. We want to know the component number of $i-$th SciFi at $j-$th layer. If the number of SciFi's are the same in each layer, we would be able to construct an integer array such that `cn=ScifiIdx(j, i)` gives the desired component number. However, constructing such an array is rather a messy task, and if the detector is modified there could be a danger in using the same procedure without modification.

The method here is simple and rather robust. As an example, take a rather complex detector consisting of 407 subdetectors as shown below.

```
1 chd1x          125 sfx            398 tasc-xyxc
2 chdx           126 sfx8           399 tasc-xyya
3 chd1y          127 sfx64          400 tasc-xyyc
4 chdy           128 sfx1sheet      401 tasc-in
5 chd            129 sfy            402 tasc-inb
6 chdmemb1       130 sfy8           403 tasc-inxa
7 chdmemb2       131 sfy64          404 tasc-inxc
8 chdassy1       132 sfy1sheet      405 tasc-inya
9 chdassy2       133 sfx1sheet      406 tasc-inyc
10 pmt-plt1      134 sfy1sheet      407 tasc
...              ...
```

The "`subdTree config` 1" command produces a hierarchy map shown in Listing 1. The numbers after | are depth of hierarchy followed by a sub-detector name or volume-shape name. If * is attached, it is a simple component without containing other components. The next number is the component number followed by the volume-shape and medium name. There are lots of SciFi's; 1 sheet of SciFi's which are position sensitive to the $x-$direction is denoted by a sub-detector `sfx1sheet` in which a number of SciFi's are contained. They are at depth 7 of the hierarchy. Another sheet of SciFi's sensitive to $y-$direction is contained by the `sfy1sheet` sub-detector.

Here, we call components which we want to identify "target". In this example, the target is SciFi. We want to distinguish SciFi's for x and y, then the minimum hierarchy specification for our "target" is 'sfx1sheet SciFi' for x and 'sfy1sheet SciFi' for y. Let's assume these are enough, i.e, other SciFi's do not get mixed. Then we may define two rank 2 (i.e, 2-D) integer arrays: name is arbitrary and we use here `SciFiXIdx` for x and `SciFiYIdx` for y.

Listing 1: hierarchy

```
H---- subdetector hierarchy ----
| 0 world         56746 sphere_w        sp
   | 1 imcbox        52531 box             sp
      | 2 imc-top          46 box             sp
         | 3 box*              1 box             Al7075
         | 3 box*              6 box             hollow
         | 3 horse*            5 horse           hollow
...
                  | 6 box*           2440 box             SCIN
                  | 6 sqtccl*        2441 sqtccl          Acrylic
                  | 6 cyl_x*         2442 cyl_x           Acrylic
         | 3 sfx1sheet      3517 box             sp
            | 4 sfx64          2598 box             sp
               | 5 sfx8           2464 box             sp
                  | 6 box*           2446 box             sp
                  | 6 sfx            2448 box             clad
                     | 7 box*           2447 box             SciFi
                  | 6 sfx            2450 box             clad
                     | 7 box*           2449 box             SciFi
                  | 6 sfx            2452 box             clad
                     | 7 box*           2451 box             SciFi
 ...
                  | 6 sfx            3511 box             clad
                     | 7 box*           3510 box             SciFi
                  | 6 sfx            3513 box             clad
                     | 7 box*           3512 box             SciFi
                  | 6 box*           3514 box             sp
         | 3 sfy1sheet      4589 box             sp
            | 4 sfy64          3670 box             sp
               | 5 sfy8           3536 box             sp
                  | 6 box*           3518 box             sp
                  | 6 sfy            3520 box             clad
                     | 7 box*           3519 box             SciFi
                  | 6 sfy            3522 box             clad
                     | 7 box*           3521 box             SciFi
 ...
                     | 7 box*           4584 box             SciFi
                  | 6 box*           4586 box             sp
         | 3 opt1x14u       6528 horse           sp
         | 3 opt1xlwr14u    6527 horse           sp
            | 4 opt1xlwr7u     6042 horse           sp
               | 5 opt1xlwr       5627 horse           sp
                  | 6 opt1xlwr1      5575 horse           sp
                     | 7 horse*         5559 horse           SciFi
                     | 7 horse*         5560 horse           SciFi
...
                     | 7 horse*        11370 horse           SciFi
                     | 7 horse*        11371 horse           SciFi
         | 3 sfx1sheet     13417 box             sp
            | 4 sfx64         12498 box             sp
               | 5 sfx8          12364 box             sp
                  | 6 box*          12346 box             sp
                  | 6 sfx           12348 box             clad
                     | 7 box*          12347 box             SciFi
...
```

A code fragment (Listing 2) will work for getting `SciFiXIdx`. Normally, we may get index arrays only once in a simulation; then the appropriate place would be in uiaev (or its slave) in ephook.f. This example subroutine `epSetIdx` is assumed to be called from uiaev (must be compiled before ephook.f is compiled). The arrays must be available for use in other places so that they are put in a module. Its name here is `modIndexing` but it is arbitrary as long as no name collision happens.

SciFiXSpec: specifies target hierarchy. One `sfx1sheet` exists in one layer and contains target SciFi's for x.

SciFiXShape: to store the shape (size) of SciFiXIdx.

ScifiXIdx: this is the target 2-D index array. The user, in principle, already knows the shape, but we use information obtained by `epCountSubdTree` and allocate it.

epCountSubdTree: may be used to get the shape (but not mandatory). The input is `SciFiXSpec`. n becomes the rank of the shape (=2).

SciFiXnum: is a 2-D pointer array. The value at an index (l,m) shows the number of targets satisfying a certain condition (see later). The size is allocated inside `epCountSubdTree` automatically. The user may normally treat it as a usual integer array. In this simple usage, a value is available only at index (1,1): SciFiXnum(1,1) shows the number of SciFiX's in each layer, that is, same as `SciFiXShape(2)`.
Besides the parameters shown in this example, there are some optional parameters (see later).

epGetIndex: the index array, `SciFiXIdx`, is obtained by calling this subroutine. The first two are input and the last three output. Optional parameters are the same as for `epCountSubdTree`.

Listing 2: Getting Index

```
      module modIndexing
!           SciFi for X
      character(len=*),parameter:: SciFiXSpec="sfx1sheet SciFi"
      integer:: SciFiXShape(2)
      integer,allocatable::  SciFiXIdx(:,:)
      integer,pointer::SciFiXnum(:,:)
      end  module modIndexing

      subroutine epSetIdx
      use modIndexing
      use modGetIndex
      implicit none
      integer:: i,j,  n
      integer:: Nlayer, NSciFi
!            get shape of SciFiXIdx.
      call epCountSubdTree(SciFiXSpec, SciFiXShape, n, SciFiXnum)
      write(0,*) 'SciFiXnum =',SciFiXnum
      Nlayer =  ScifiXShape(1)
      NSciFi =  ScifiXShape(2)
      write(0,*) ' ScifiXShape=', Nlayer, NSciFi
      allocate( SciFiXIdx(Nlayer, NSciFi) )

      call epGetIndex(SciFiXSpec, SciFiXShape, SciFiXIdx, SciFiXnum)

      do i = 1, Nlayer
         do j = 1, NSciFi
            write(0,'(a, i4, a, i4, a, i6)')
```

```
28      *       ' ScifiX # ',j, ' in layer ',i, ' has  comp#=',
29      *      SciFiXIdx(i,j)
30          enddo
31       enddo
32       end subroutine epSetIdx
```

Once SciFiXIdx is obtained, the user can get the component number of a SciFi specified by its layer (1:Nlayer) and the number (1:NSciFi). The output from the example will look like,

```
 ScifiX #    1 in layer    1 has  comp#=  2447
 ScifiX #    2 in layer    1 has  comp#=  2449
 ScifiX #    3 in layer    1 has  comp#=  2451
...
```

3.1 A minor problem

The SciFiXIdx has a "top-down" hierarchy or Japanese style addressing, i.e, indexes appear with the higher (less deeper) hierarchy order. This is suited for the C-language style memory allocation. In many cases, some consecutive numbers of SciFi's at a layer may be accessed in the program at a time. These consecutive SciFi indexes are allocated by the C-language consecutively in the memory space but by Fortran it is sparse and could lead to delay of the memory access.

This effect is expected normally negligible but if the user wants to European/American style addressing suited for Fortran memory allocation, the user may re-shape the SciFiIdx as follows.

```
1   integer:: FSciFiXShape(2)
2   integer,allocatable::  FSciFiXIdx(:,:)
3 ...
4   forall(i=1:2) FSciFiXShape(3-i) = SciFiXShape(i)
5   allocate( FSciFiXIdx(NSciFi, Nlayer) )
6   FSciFiXIdx   =reshape(SciFiXIdx, FSciFiXShape, order=(/2,1/))
7   deallocate(SciFiXIdx  )! if no more needed, delete.
```

Then, the first index of FSciFiXIdx is for the SciFi numbers (1:NSciFi) and the second the layer number (1:Nlayer). The same method can be applied for higher rank arrays.

3.2 Combining two or more index arrays

In a similar fashion, one can make SciFiYIdx. If the number of x and y SciFi's are the same, one may want to use unified name such as SciFiIdx and distinguish x and y by index but not by the array name. For that, we have to define a rank 3 (3-D) array SciFiIdx(2,...). The coding will be like this.

```
1    integer:: SciFiShape(3)
2    integer,allocatable::  SciFiIdx(:,:,:)
3    ...
4 !      combine two
5    SciFiShape(1) = 2
6    SciFiShape(2:3) = SciFiXShape(1:2)
7    allocate( SciFiIdx(2, Nlayer, NScifi) )
8    SciFiIdx(1,:,:) = SciFiXIdx(:,:)
9    SciFiIdx(2,:,:) = SciFiYIdx(:,:)
10
11   deallocate( SciFiXIdx )  ! delete if no more needed
12   deallocate( SciFiYIdx )
```

Adding one more array, say, `SciFiZIdx`, is simply done by changing a part and adding some like

```
SciFiShape(1) = 3
allocate( SciFiIdx(3, Nlayer, NScifi) )
SciFiIdx(3,:,:) = SciFiZIdx(:,:)
```

Fortran oriented arrays are also similarly combined or combined `SciFiIdx` can be re-shaped to a Fortran oriented array.

3.3 Making combined index directly

A direct way of making a combined index array like `SciFiIdx` above, is to give a hierarchy like "`|sfx1sheet,sfy1sheet| SciFi`". The code fragment is

```
      module modIndexing
!          SciFi for X,Y
      character(len=*),parameter::SciFiSpec=
     *         "|sfx1sheet,sfy1sheet|  SciFi"
      integer:: SciFiShape(3)   !  if | | is used, rank must be +1
      integer,allocatable::  SciFiIdx(:,:,:)
      integer,pointer::SciFinum(:,:)
!         reshaped index; Fortran oriented version
      integer:: FSciFiShape(3)
      integer,allocatable::  FSciFiIdx(:,:,:)
      end  module modIndexing

      subroutine epSetIdx
      use modIndexing
      use modGetIndex
      implicit none
      integer:: i,j,k, n
      integer:: Nlayer, NSciFi
      character(len=1):: xy(2)=(/'X','Y'/)

      call epCountSubdTree(SciFiSpec, SciFiShape, n, SciFinum)
      write(0,*) 'SciFinum =',SciFinum
      Nlayer =  ScifiShape(2)
      NSciFi =  ScifiShape(3)

      write(0,*) ' ScifiShape=', SciFiShape(1:3)

      allocate( SciFiIdx(2, Nlayer, NSciFi) )
      call epGetIndex(SciFiSpec, SciFiShape, SciFiIdx, SciFinum)
      do k = 1, 2
         do j = 1, Nlayer
            do i = 1, NSciFi
                write(0,'(a,a, i4, a, i4, a, i6)')
     *         ' Scifi ',xy(k),'  # ',i, ' in layer ',
     *         j, ' has  comp#=',  SciFiIdx(k, j,i)
         enddo
      enddo
!           reshape; Fortran oriented array
      forall(i=1:3) FSciFiShape(4-i) =SciFiShape(i)
      allocate( FSciFiIdx(NSciFi, Nlayer, 2) )
      FSciFiIdx =
```

```
      *      reshape(SciFiIdx, FSciFiShape, order=(/3,2,1/))
       do k = 1, 2
          do j = 1, Nlayer
             do i = 1, NSciFi
               write(0,'(a, i4, a, i4, a, i6)')
      *        'F Scifi ',xy(k), ' # ',i , ' in layer ',j,
      *        ' has  comp#=',  FSciFiIdx(i,j,k)
              enddo
          enddo
       enddo
       end subroutine epSetIdx
```

The restriction and note on `|abc,xyz|` notation.

- Lets regard `|...|` in the "spec"ification of hierarchy 1 item. It can be used only for the first item in the "spec" data (if there is only 1 item, it must be the target, then, we cannot use this).
- The rank of "shape" and "index" array is usually the number of items in the "spec" data. But if there is `|...|`, 1 must be added.
- Items inside `|...|` must be "," separated. At least 2 items must exist there (if only 1, `||` effect is neglected). The maximum number of items there must be <= 4. Blanks may be placed inside `|...|`. So `|abc,xyz|`, `| abc, xyz|`, `|abc , xyz |`, `|abc , xyz,pqr|`, etc are OK.
- Suppose hierarchy of `spec="|A1,A2,A3| B T"`. In this case, there are 3 paths "A1 B T", "A2 B T" and "A3 B T". Let's call `|...|` part top branch. Top branch counter (say, name it *tbc*) runs from 1 to 3 in this case. The "num" pointer array is defined as num(3,1) automatically; 3 comes from the number of items in `|...|`. Its value (3 in this case) is contained in a variable, ntbr. num(ibc,1) becomes the number of target "T" in the *ibc*-th branch. If no top branch is specified, we may regard that ntbr=1.

3.4 Time needed for getting the index array

In the example so far, the number of components is large; reading and processing the "config" file before starting particle tracking simulation inside the detector takes ~63 s. So one may feel nervous about time needed for getting the index array. However, it takes only ~0.15 s to get 2 index arrays, convert them to Fortran oriented shape, combine them and, print out ~ 14500 lines (By MacBook pro 2.9GHz).

In many smaller scale config's, time spent for getting an index array is <ms.

4 Treating odd sub-detectors

Suppose a detector, as in Fig..1(left), consisting of 2 layers with the same structure[3]. This detector has odd structure; "sheet" has a number of subdetectors and each of which has some deeper structure and finally target CHDx's (in total 100; CHDx is an alias of SCIN here). Another CHDx container in the same layer, "oddx4", is odd; its size and the number of CHDx's (=12) inside are different from those in "sheet". So if the user makes two index arrays for spec="sheet CHDx" and for spec="oddx4 CHDx", these CHDx's can be handled.

However, in some case, the user may want to define one index array and treat all of the CHDx's by that array. This is possible a) if the number of sheet's is the same as that of oddx4's, and b) they are last but one items in the "spec" data.

[3] "Config" files in this section are available in UserHook/Indexing.

The “sheet” and “oddx4” satisfy this condition. Then, we can use the following “spec” data: `spec="(sheet,oddx4)  CHDx"`. The usage of “,” and space inside ”()“ is the same as the top branch case. But c) the number of items inside `(...)` must be 2. Therefore, if there is another odd subdetector containing CHD’s in the same layer (say, aoddx4, Fig.(right)), we cannot use

`"(sheet,oddx4,aoddx4)  CHDx"`

(in this case, error stop will happen: violation of c)).

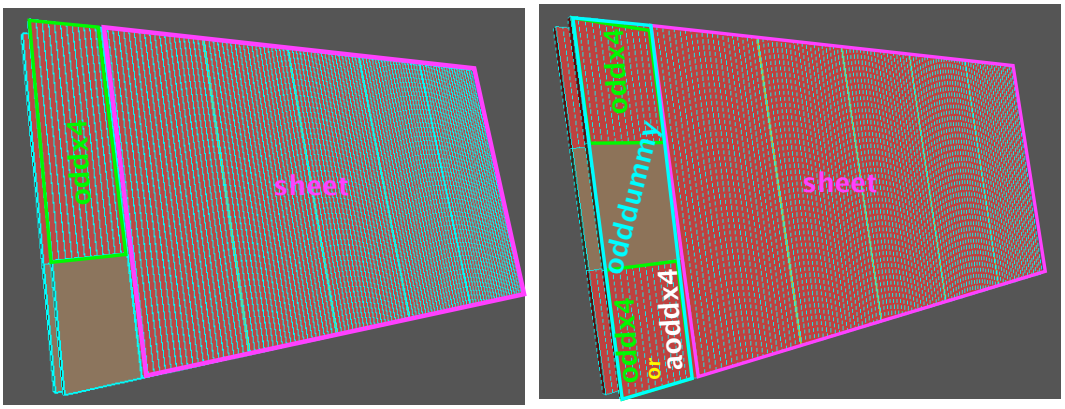

Figure 1: Left: Odd sub-detectors are placed at left-hand side. (oddx4).

Even if one uses spec= `"(sheet,oddx4)  CHDx"`, if there is one more “oddx4” (instead of aoddx4) as in Fig.(right), the number of oddx4’s becomes 2 while that of sheet’s is 1 in the same layer, so a) is violated (in this case, no error stop will happen, but index will look strange).

Listing 3: Treating odd structure

```
!          configB is used;  to be specified in FirstInput
      module modIndexing
      character(len=*),parameter::  Spec="(sheet,oddx4) CHDx"
!        next Spec  can be used without changing other part
!           Spec="(oddx4,sheet)  CHDx"
      integer:: Shape(2)
      integer,allocatable::  CHDIdx(:,:)

      integer:: FShape(2)
      integer,allocatable::  FCHDIdx(:,:)

      integer,pointer:: CHDnum(:,:)
      end       module modIndexing

      subroutine epSetIdx
      use modIndexing
      use modGetIndex
      implicit none
      integer i,j,  n, oddl
      integer Nlayer, NCHDx

      call epCountSubdTree(Spec, Shape, n, CHDnum)
      write(0,*) '1 num=', CHDnum(1,:)
      write(0,*) 'Shape=', Shape(:)

      oddl = index(Spec, "oddx4")
      if( oddl > 3 ) then
         write(0,*) 'Odd part contains',CHDnum(1,2), ' CHD,'
```

```
29          write(0,*) 'they are located at the last part'
30       else
31          write(0,*) 'Odd part contains', CHDnum(1,1), ' CHD,'
32          write(0,*) 'they are located at the fisrt part'
33       endif
34       Nlayer = Shape(1)
35       NCHDx  = Shape(2)
36       write(0,*) ' Nlayer=',Nlayer, ' NCHDx=',NCHDx
37       allocate( CHDIdx(Nlayer, NCHDx) )
38
39       call epGetIndex(Spec, Shape, CHDIdx, CHDnum)
40
41       do i = 1, Nlayer
42          do j = 1, NCHDx
43             write(0,'(a, i4, a, i4, a, i6)')
44      *       ' CHD # ',j, ' in layer ',i, ' has  comp#=',
45      *       CHDIdx(i,j)
46          enddo
47       enddo
48       end  subroutine epSetIdx
```

In this case, num(1,1) becomes the number of targets in "sheet" (100), and num(1,2) that in "oddx4" (12). In general, num(*,2) is used to count the number of targets in the "odd" subdetector. Shape(1) becomes the number of layers (=2) and Shape(2) becomes the total number of targets in each layer, i.e, num(1,1) + num(1,2). In CHDIdx(*,1:100) is for "sheet" and CHDIdx(*,101:112) for "oddx4".

Which is odd, "Sheet" or "oddx4" ? This is arbitrary. So `Spec="(oddx4,sheet) CHDx"` may be used, instead of `Spec="(sheet,oddx4) CHDx"`. In this case, the roll of num(1,1) and num(1,2) is interchanged.

4.1 How to overcome a) and c)

If the user insists on using only 1 index array, one workaround for the case of Fig.(right) is to put every geometrically related subdetectors in a larger subdetector (odddummy in the right Fig.), and use `Spec="(sheet,odddummy) CHDx"`. In this case, num(1,1)=100 and num(1,2)=24 will result.

4.2 How about 1 layer lacks an odd subdetector ?

If "odddummy" is missing in one of the 2 layers, or more generally, suppose the number of layers is much larger than 2, and some of them lack an "odddummy", what will happen ? This violates condition a) again. So not permitted, but no error will be reported. If the problematic layer is the last one, the index array will be created correctly: index values at missing layers will be 0. However, this type of usage is not recommended.

4.3 If top branch is applied ?

In the previous example, "sheet" and "odddummy" (or "oddx4") are regarded as odd subdetector case. They are, however, at the top hierarchy so we may regard them a top branch by putting `|...|`. That is, what happens if we use `Spec="|sheet,odddummy| CHDx"` ?

In this case, `Shape` becomes (2,2,100) (i.e, rank becomes 3) and CHDnum(1,1) = 100, CHDnum(2,1) = 24. The shape of CHDIdx becomes (2,2,100).

CHDIdx(1,1,1:100) are for 100 CHDx's in layer 1 of "sheet".

CHDIdx(1,2,1:100) are for 100 CHDx's in layer 2 of "sheet".

CHDIdx(2,1,1:24) are for 24 CHDx's in layer 1 of "odddummy".

CHDIdx(2,2,1:24) are for 24 CHDx's in layer 2 of "odddummy".

The memory space of CHDIdx(2, 1:2, 24:100) are not used and 0 is filled.

If some "odddummy" is missing as in Sec.4.2, the result will be also NG, except for the case that the layer is the last one.

5 Optional parameters

Subroutines `epCountSubdTree` and `epGetIndex` can take same optional parameters. They can be specified by using the following names:

target: Normally the target is assumed to be a simple component (i.e, not containing another component) and is a medium name.
If this is not enough, one may use
`call epCountSubdTree(..., target="xxx")`
or
`call epGetIndex(..., target="xxx")`
where "..." denotes mandatory parameters[4].
The number and order of optional parameters are arbitrary. xxx may take one of

container: The target is container of another component

any: The target may be simple or container.

simple: This is default so need not be given, but is acceptable.

subd: The target name is not a medium name but is subdetector name.

dE: This is used as dE=2 etc in the optional parameter position. If the target component's countDE (energy loss count specification) does not match with this dE value, the target is not accepted. Normal value of dE is `{2,1,0,-1,-2}`.

IO: This is to check the matching with countIO, i.e, (particle in/out count specification). Normally value is IO=`{1,2,3}`.
If both "dE" and "IO" are specified, "or" is taken. Requirements by other parameters, if any, are processed as "and".

judgeBy: This is different from others since this specifies a user supplied logical function which judges that the current target candidate is to be accepted or not. Therefore, this may be prepared only if other target specification does not work well. The program structure will be like Listing 4.

Listing 4: Ultimate judge

```
1       subroutine   epSetIdx  ! assumed to be called from uiaev.
2       implicit none
3       logical, external:: ultimante
4       ...
5       call  epCountSubdTree( ...   judgeby=ultimate)
6       ...
7       call  epGetIndex( ...   judgeby=ultimate)
8       ...
9       end subroutine epSetIdx
```

[4]The same optional parameters must be given to epCountSubdTree and epGetIndex for the same "spec" data.

```
10
11        function ultimate(mother, cn) result(ans)
12        use modGetIndex
13        implicit none
14  #include "Zep3Vec.h"
15  #include "Zcnfig.h"
16        integer,intent(in)::mother ! subdetector index of the mother
17                  !  of cn.  probably not needed.
18        integer,intent(in):: cn ! comp. # of the target candidate
19        logical:: ans
20          ...
21        ans = ... (give .true. or .false.; .ture.=> accptable.)
22        end  function ultimate
```

5.1 Some examples

We see "clad" in subdetector hierarchy shown in Listing 1. It is an alias of "SciFi". Therefore, if we didn't use the alias, "clad" appeared as "SciFi" like:

```
1          ...
2                     | 6 sqtccl*         2441 sqtccl           Acrylic
3                     | 6 cyl_x*          2442 cyl_x            Acrylic
4            | 3 sfx1sheet       3517 box              sp
5               | 4 sfx64           2598 box              sp
6                  | 5 sfx8            2464 box              sp
7                     | 6 box*            2446 box              sp
8                     | 6 sfx             2448 box              SciFi
9                        | 7 box*            2447 box              SciFi
10                    | 6 sfx             2450 box              SciFi
11                       | 7 box*            2449 box              SciFi
```

Even with this condition, `spec="sfx1sheet SciFi"` is enough for selecting target "SciFi" at depth 7 since `target="simple"` is default. In some case, one may want to make "SciFi" at depth 6 the target. In this case, it is not recognized as matching with the target by the "spec" shown above. To make it work, we may give (..., `target="container"`) in the calling sequence of epCountSubdTree and epGetIndex. If we want to make "SciFi" at depth 6 and 7 the target simultaneously, we may give (..., `target="any"`).

It is possible to select "sfx" at depth 6 by giving `spec="sfx1sheet sfx"`. In this case, "sfx" is not media name but is subdetector name so that we have to give (..., `target="subd"`).

6 Complex case

If "spec" is like `"|A1,A2|  (C1,C2) T"` or `"|A1,A2| B (C1,C2) T"`, what will happen ? This complex case hasn't been tested, since no such complex case has been met yet. As a program logic, this type should be treated properly. But no test yet!

7 Utility

By issuing a command[5], `tree2index`, the user can test some class of "spec". The usage will be shown simply by typing `tree2index`. If the user forgets the name, `configMenu` command will show it since this is related to "config". The usage is:

[5]The `PATH` must be set properly, i.e, $EPICSTOP/Scrpt must be included there.

`tree2index` config spec

where “config” is the path to a “config” file, “spec” a specification of a target by hierarchy such as

sfx1sheet SciFi

’|sfx1sheet,sfy1sheet|’ SciFi

’(sheet,oddx4)’ CHDx

(...) or |...| must be enclosed by commas. The max number of items in “spec” is 4. (If |...| is used, 3). Optional parameters are not permitted. For such a case, visit `$EPICSTOP/UserHook/Indexing`.

The output is on sdtout so it may be re-directed to a file like, `tree2index` config spec > testout