BUILDING A DETECTOR IN FAIRSHIP

Annarita Buonaura
Università di Napoli & INFN

6th SHiP Collaboration Meeting
CERN
October 7 - October 9, 2015

OUTLINE

- Introduction

- Creating a new detector class
  - The CMakeLists file
  - The Hits class
  - The Detector Class
    - ConstructGeometry()
    - ProcessHits()
    - AddHit()

- How to make everything work
  - The LinkDef.h file
  - Make FairShip know about your detector

INTRODUCTION

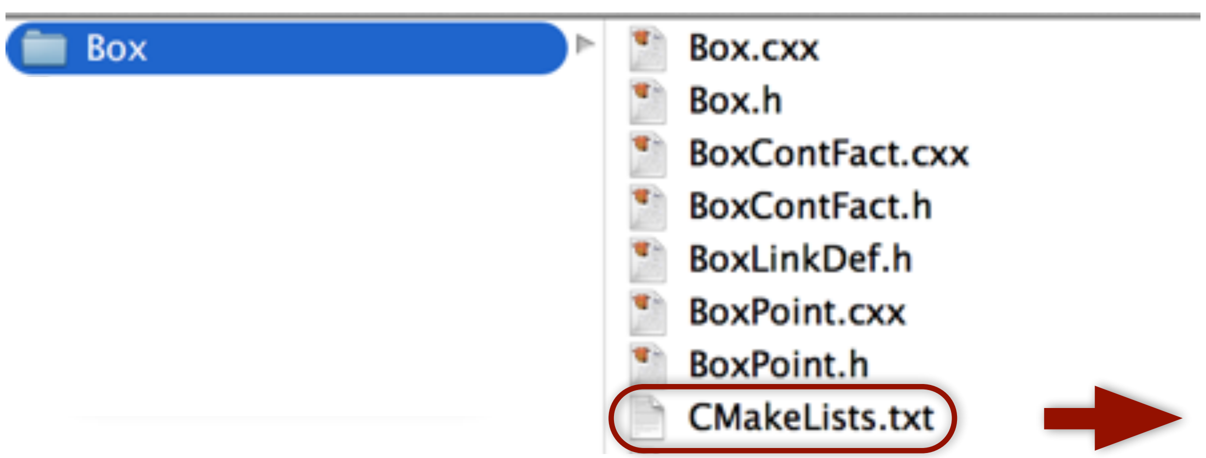
- SHiP geometry environment is mainly based on the **ROOT/TGEO package**.
  - It is a tool for building, browsing, navigating and visualizing detector geometries
  - Particle transport is obtained by working in correlation with simulation packages such as GEANT3, GEANT4 and FLUKA

- To create a new detector you have to implement some classes which will describe your detector.
  - To describe a detector (possibly in a new folder of FairShip/) it is important to implement:
    - the CMakeFile
    - the detector class
    - the detector MC Point class

CREATING A NEW DETECTOR CLASS

THE CMakeLists FILE

- For a standalone detector create a new folder.
- When creating a new folder (e.g: FairShip/Box) it is necessary to first define a CMakeLists file containing the names of the .cxx file in the folder.
- It will create a library which includes the source files written in the folder .




```
set(INCLUDE_DIRECTORIES
${BASE_INCLUDE_DIRECTORIES}
${CMAKE_SOURCE_DIR}/shipdata
${CMAKE_SOURCE_DIR}/Box
${ROOT_INCLUDE_DIR}

)

include_directories( ${INCLUDE_DIRECTORIES})
include_directories(SYSTEM ${SYSTEM_INCLUDE_DIRECTORIES})

set(LINK_DIRECTORIES
${ROOT_LIBRARY_DIR}
${FAIRROOT_LIBRARY_DIR}

)

link_directories( ${LINK_DIRECTORIES})

set(SRCS
Box.cxx
BoxPoint.cxx
BoxContFact.cxx
)

Set(HEADERS )
Set(LINKDEF BoxLinkDef.h)
Set(LIBRARY_NAME Box)
Set(DEPENDENCIES Base ShipData GeoBase ParBase Geom Cint Core)

GENERATE_LIBRARY()
```

THE HITS CLASS

```
class BoxPoint : public FairMCPoint
{

  public:

    /** Default constructor **/
    BoxPoint();


    /** Constructor with arguments
     *@param trackID  Index of MCTrack
     *@param detID    Detector ID
     *@param pos      Ccoordinates at entrance to active volume [cm]
     *@param mom      Momentum of track at entrance [GeV]
     *@param tof      Time since event start [ns]
     *@param length   Track length since creation [cm]
     *@param eLoss    Energy deposit [GeV]
     **/


    BoxPoint(Int_t trackID, Int_t detID, TVector3 pos, TVector3 mom,
                Double_t tof, Double_t length, Double_t eLoss, Int_t pdgCode);

    /** Destructor **/
    virtual ~BoxPoint();

    /** Output to screen **/
    virtual void Print(const Option_t* opt) const;


    Int_t PdgCode() const {return fPdgCode;}

  private:


    Int_t fPdgCode;

    /** Copy constructor **/

    BoxPoint(const BoxPoint& point);
    BoxPoint operator=(const BoxPoint& point);

    ClassDef(BoxPoint,1)

};
```

BoxPoint.h

Functions defined in Box.cxx

THE DETECTOR CLASS

```
class Box : public FairDetector
{
public:
    Box(const char* name, const Double_t BX, const Double_t BY, const Double_t BZ, Bool_t Active, const char* Title = "Box");
    Box();
    virtual ~Box();

    /**      Create the detector geometry        */
    void ConstructGeometry();

    /**      Initialization of the detector is done here    */
    virtual void Initialize();

    /**  Method called for each step during simulation (see FairMCApplication::Stepping()) */
    virtual Bool_t ProcessHits( FairVolume* v=0);

    /**       Registers the produced collections in FAIRRootManager.     */
    virtual void   Register();

    /** Gets the produced collections */
    virtual TClonesArray* GetCollection(Int_t iColl) const ;

    /**      has to be called after each event to reset the containers      */
    virtual void   Reset();

    /**      How to add your own point of type BoxPoint to the clones array */

    BoxPoint* AddHit(Int_t trackID, Int_t detID, TVector3 pos, TVector3 mom,
                     Double_t time, Double_t length, Double_t eLoss, Int_t pdgCode);


    virtual void   CopyClones( TClonesArray* cl1,  TClonesArray* cl2 , Int_t offset) {;}
    virtual void   SetSpecialPhysicsCuts() {;}
    virtual void   EndOfEvent();
    virtual void   FinishPrimary() {;}
    virtual void   FinishRun() {;}
    virtual void   BeginPrimary() {;}
    virtual void   PostTrack() {;}
    virtual void   PreTrack() {;}
    virtual void   BeginEvent() {;}
```

Box.h

```
    Box(const Box&);
    Box& operator=(const Box&);

    ClassDef(Box,1)

private:

    /** Track information to be stored until the track leaves the active volume. */
    Int_t           fTrackID;          //!  track index
    Int_t           fVolumeID;         //!  volume id
    TLorentzVector  fPos;              //!  position at entrance
    TLorentzVector  fMom;              //!  momentum at entrance
    Double32_t      fTime;             //!  time
    Double32_t      fLength;           //!  length
    Double32_t      fELoss;            //!  energy loss

    /** container for data points */
    TClonesArray*   fBoxPointCollection;

protected:

    Double_t    BoxX;
    Double_t    BoxY;
    Double_t    BoxZ;

    Int_t InitMedium(const char* name);

};
```

Functions defined in Box.cxx

ConstructGeometry()

```
void Box::ConstructGeometry()
{
    InitMedium("Scintillator");
    TGeoMedium *scint =gGeoManager->GetMedium("Scintillator");


    TGeoVolume *topDV= gGeoManager->GetVolume("DecayVolume");
    TGeoBBox *BOX1 = new TGeoBBox(BoxX/2,BoxY/2,BoxZ/2);
    TGeoVolume *VBOX1 = new TGeoVolume("volBox", BOX1,scint);
    VBOX1->SetLineColor(kRed);
    topDV->AddNode(VBOX1,1,new TGeoTranslation(0,0,50));
    AddSensitiveVolume(VBOX1);
}
```

CREATING THE SHAPE

- The basic bricks for building-up the model are called ***volumes***.
- Volumes are put one inside another making an in-depth hierarchy. The biggest one containing all others defines the "***world***" of the model.
- Since in FairShip the world has already been defined, when writing a new detector class it can be called through:

```
TGeoVolume *top = gGeoManager->GetTopVolume();
```

NB: The default units are in centimeters

- Each volume has a **shape**. The shape provides the definition of the l o c a l coordinate system of the volume.
- Any volume must have a shape.
- Any shape has to derive from the base TGeoShape class.
- At the moment a set of 20 basic shapes is provided (***primitive***) for example: BOX , TRAP, TUBE, CONE, SPHE, ... but there is also the possibility to create shapes as a result of Boolean operations between primitives. These are called ***composite shapes***

CREATING THE SHAPE (II)

- All primitives have constructors like:

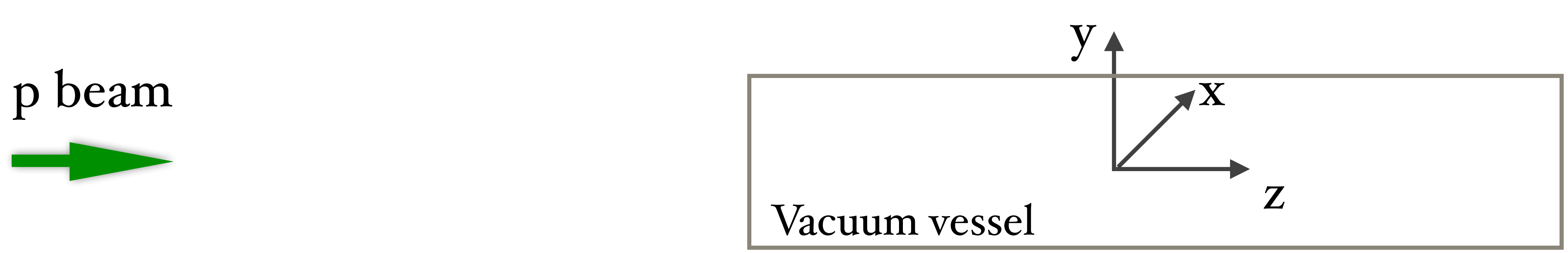
TGeoXXX(const char *name, <type> param1, <type> param2, ...);
TGeoXXX(<type> param1, <type> param2, ...);

Example

In YourDetector.cxx file, when constructing the geometry:

```
TGeoBBox *Box = new TGeoBBox(Double_t dx,Double_t dy,Double_t dz);
```

NB: dx, dy and dz represent the half-lengths on X, Y and Z axis



DEFINING THE MEDIA

- Together with shapes, volumes need media to be created, because materials represent the physical properties of the solid from which a volume is made.
- The *TGeoMedium class* defines the **media**, that are material with tracking parameters needed for the transport (sensitivity flag, field flag, max field value ….)
- In FairShip media are read by the geometry/media.geo file throughout the private function ***InitMedium***:

```
Int_t XXX::InitMedium(const char* name)
{
   static FairGeoLoader *geoLoad=FairGeoLoader::Instance();
   static FairGeoInterface *geoFace=geoLoad->getGeoInterface();
   static FairGeoMedia *media=geoFace->getMedia();
   static FairGeoBuilder *geoBuild=geoLoad->getGeoBuilder();
   FairGeoMedium *ShipMedium=media->getMedium(name);
   if (!ShipMedium)
   { Fatal("InitMedium","Material %s not defined in media file.", name);
     return -1111;}
   TGeoMedium* medium=gGeoManager->GetMedium(name);
   if (medium!=NULL)
      return ShipMedium->getMediumIndex();
   return geoBuild->createMedium(ShipMedium);
}
```

DEFINING THE MEDIA

- Together with shapes, volumes need media to be created, because materials represent the physical properties of the solid from which a volume is made.
- The *TGeoMedium class* defines the **media**, that are material with tracking parameters needed for the transport (sensitivity flag, field flag, max field value ….)
- In FairShip media are read by the **geometry/media.geo** file throughout the private function ***InitMedium***:

```
Int_t XXX::InitMedium(const char* name)
{
   static FairGeoLoader *geoLoad=FairGeoLoader::Instance();
   static FairGeoInterface *g
   static FairGeoMedia *med
   static FairGeoBuilder *ge
   FairGeoMedium *ShipMe
   if (!ShipMedium)
   { Fatal("InitMedium","Materi
     return -1111;}
   TGeoMedium* medium=g
   if (medium!=NULL)
     return ShipMedium->getMediumIndex();
   return geoBuild->createMedium(ShipMedium);
}
```

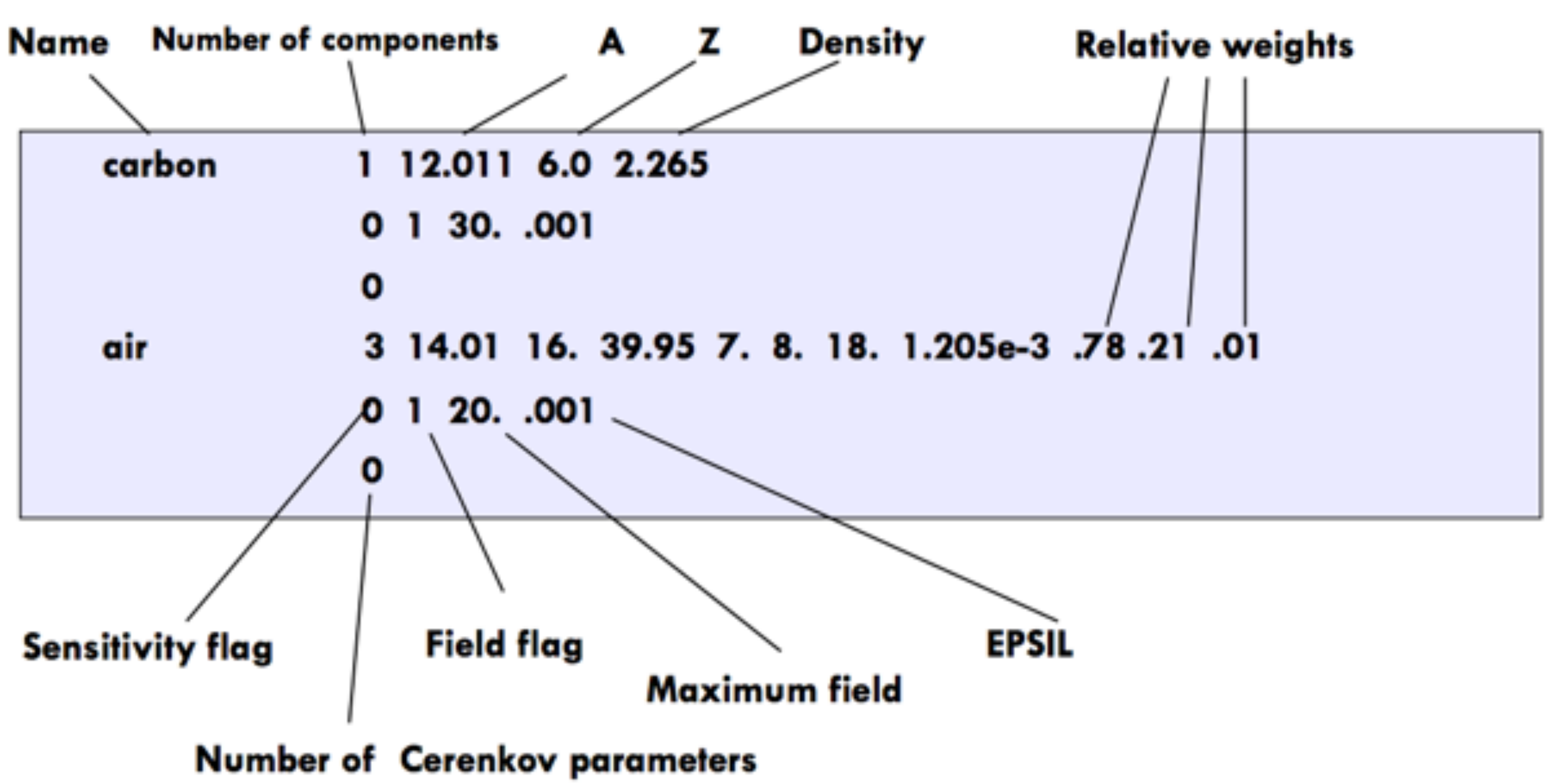
Example

In YourDetector.cxx file, when constructing the geometry:

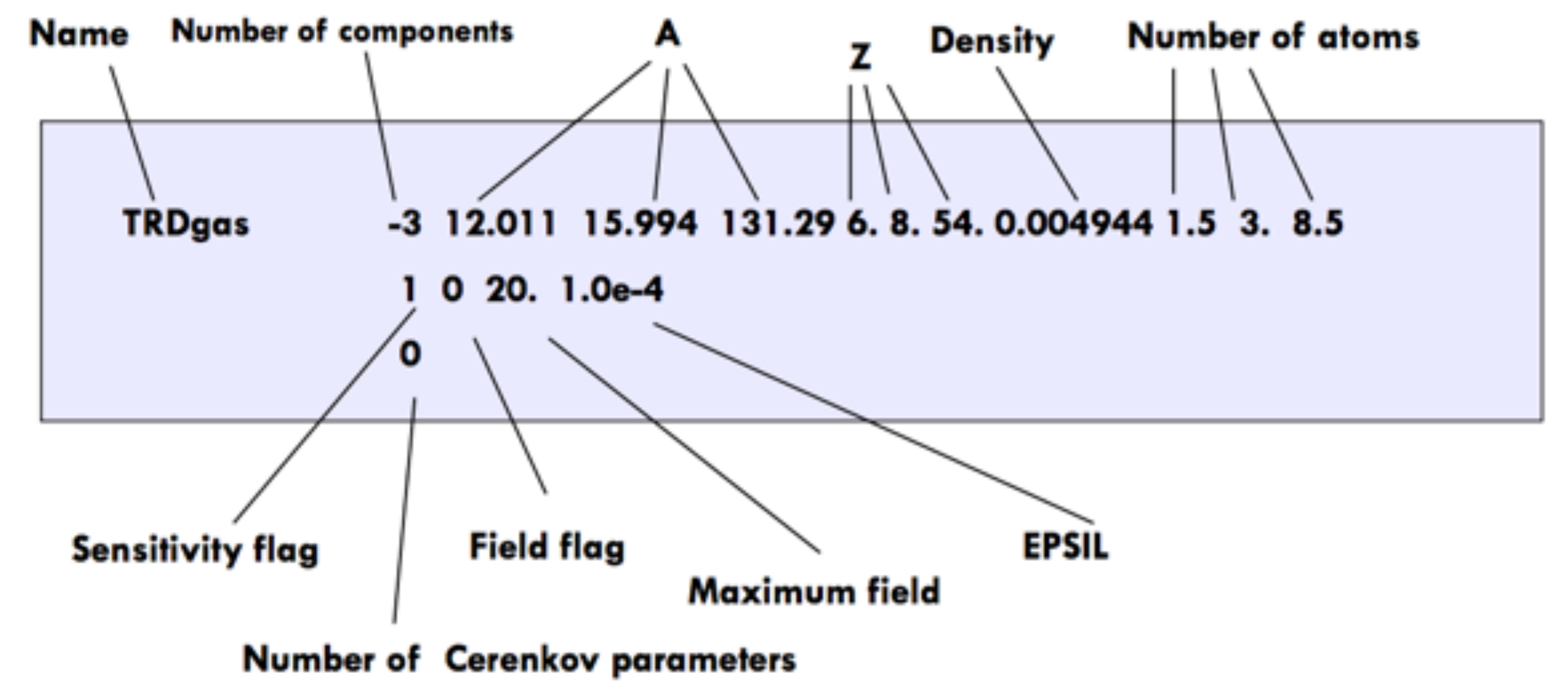
```
InitMedium("iron");
TGeoMedium *Fe =gGeoManager->GetMedium("iron");
```

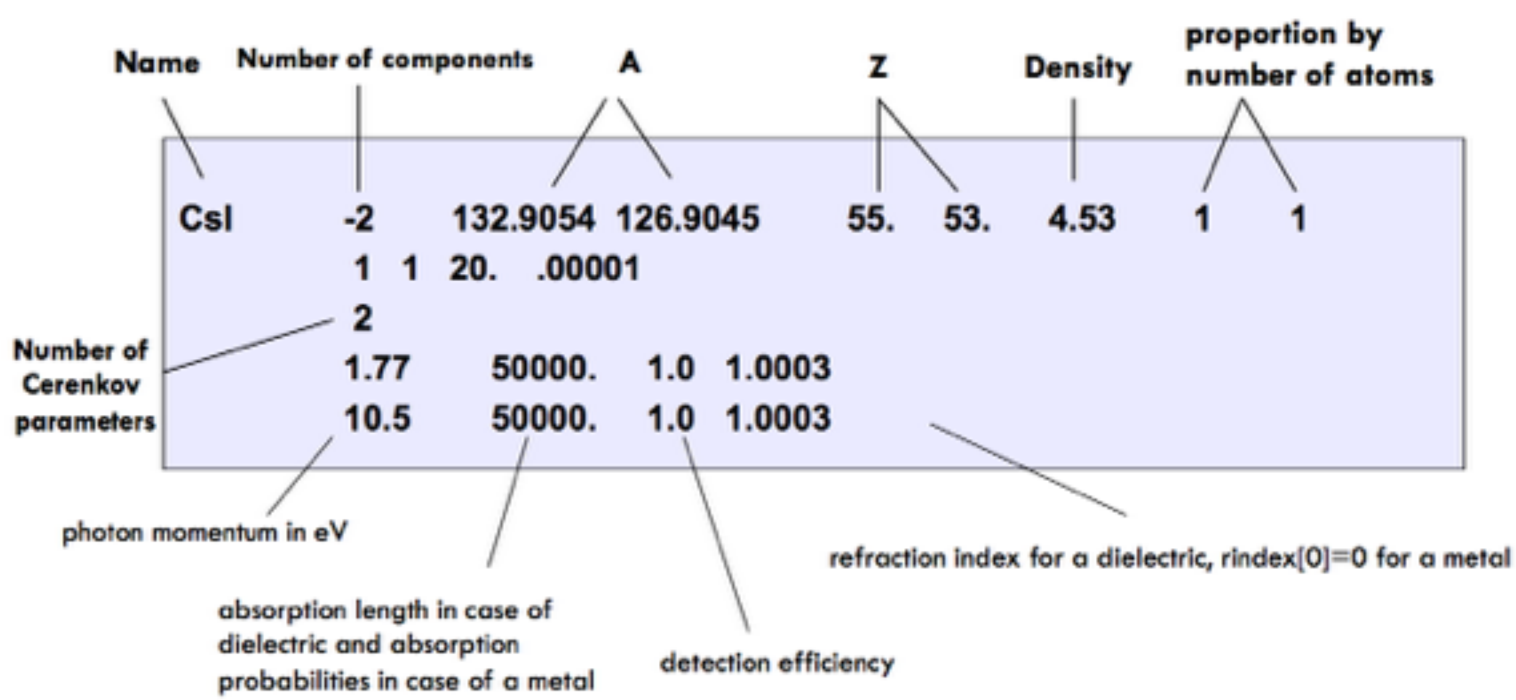
DEFINING THE MEDIA (II)

- New media can be added to the geometry/media.geo file.
- There can be multiple kind of definitions according to the knowledge of the different properties of the considered medium



DEFINING THE MEDIA (III)





CREATING THE VOLUME

- Volumes need media and shapes in order to be created.
- Both containers and contained volumes must be created before linking them together, and the relative transformation matrix must be provided.
- Any volume has to be positioned somewhere otherwise it will not be considered as part of the geometry.

```
// Making a volume out of a shape and a medium.
TGeoVolume *vol = new TGeoVolume("VNAME",ptrShape,ptrMed);
```

Example

In YourDetector.cxx file, when constructing the geometry:

TGeoVolume *VBox = **new TGeoVolume**("volBox", Box, scint);

POSITIONING THE VOLUME

- In volume creation no need to specify whether it contains or not other volumes.
- Adding daughters to a volume implies creating those and adding them one by one to the list of daughters.
- Positions of daughter volumes with respect to the center of mother volume must be known, hence it is necessary to supply a geometrical transformation when positioning daughter volumes.
- Daughter volumes must not extrude the mother shape.
- Volumes positioned in the same container must not overlap with each other

TGeoVolume::AddNode(TGeoVolume *daughter,Int_t usernumber, TGeoMatrix *matrix)

Example

In YourDetector.cxx file, when constructing the geometry:

```
Int_t nReplica = 1;
TGeoTranslation *t = new TGeoTranslation(tx,ty,tz);
top -> AddNode(VBox, nReplica, t);
```

Mother Volume (top)

Number of Replica (nReplica)

Translations along x,y,z wrt center of mother volume (ty)

POSITIONING THE VOLUME (II)

- If the detector consists of a repetition of unitary cells (e.g. 10 iron layers), it is important not to create a different shape and a different volume for each cell.
- It is enough to replicate the ones that have been already created

Example

In YourDetector.cxx file, when constructing the geometry:

```
Int_t nReplica = 10;
TGeoBBox *sbox = new TGeoBBox(dx1, dy1, dz1);
TGeoVolume *Vsbox = new TGeoVolume("volIron", sbox, iron);
for(Int_t n =0; n< nReplica; n++)
{
      Double_t t'z = n*0.1;
      TGeoTranslation *t = new TGeoTranslation(0,0,t'z);
      Box -> AddNode(VBox, n, t);
}
```

ACTIVE VS PASSIVE VOLUMES

- ***Passive volumes*** should inherit from the FairModule class.

Example

In YourPassiveDetector.cxx file, when defining the class constructor:

PBox::**PBox**()::**FairModule**("PassiveBox", "Title")

ACTIVE VS PASSIVE VOLUMES

- ***Passive volumes*** should inherit from the FairModule class.
- ***Active volumes*** must inherit from the FairDetector class.
- The Detector class is a sub class of the module which implements extra functions called from the event loop of the MC to make some actions during simulation

Example

In YourActiveDetector.cxx file, when defining the class constructor:

ABox::**ABox**()::**FairDetector**("ActiveBox", "Title",kTRUE)

Bool_t IsActive

- Sensitive volumes defined using *AddSensitiveVolume(volName)*

Example

In YourActiveDetector.cxx file, when constructing the geometry:

AddSensitiveVolume(VBox);

MAGNETIC FIELD

- The value of the magnetic field can be defined as a private member of the detector class.
- Then in YourDetector.cxx file, when constructing geometry:

Example

```
TGeoUniformMagField *magField = new TGeoUniformMagField(0.,-MagneticField,0.);

volBox->SetField(magField);
```

Note: This is valid only in FairShip.
Necessary to manipulate G4 geometry to enable magnetic field in active shielding. Private fix in run_simScript.py to make it work

```
# manipulate G4 geometry to enable magnetic field in active shielding, VMC can't do it.
import geomGeant4
geomGeant4.setMagnetField() # ('dump') for printout of mag fields
```

PARAMETER FILES

- In order to study different detector designs, basic geometry parameters should be given by instantiation of the geometry objects, not hardcoded in C++ class.
  - Basic parameters are in **geometry/geometry_config.py**

Example

```
c.Box = AttrDict(z=0*u.cm)
c.Box.BX = 3*u.m;
c.Box.BY = 3*u.m;
c.Box.BZ = 3*u.m;
```

  - Geometry objects are created by **python/shipDet_conf.py** and declared to the run manager FairRunSim()

Example

```
Box = ROOT.Box("Box",ship_geo.Box.BX, ship_geo.Box.BY,
ship_geo.Box.BZ, ROOT.kTRUE)
run.AddModule(Box)
```

PARAMETER FILES

- In order to study different detector designs, basic geometry parameters should be given by instantiation of the geometry objects, not hardcoded in C++ class.
  - Basic parameters are in **geometry/geometry_config.py**

Example

```
c.Box = AttrDict(z=0*u.cm)
c.Box.BX = 3*u.m;
c.Box.BY = 3*u.m;
c.Box.BZ = 3*u.m;
```

- Geometry objects are created by **python/shipDet_conf.py** and declared to the run manager FairRunSim()

Example

```
Box = ROOT.Box("Box",ship_geo.Box.BX, ship_geo.Box.BY,
ship_geo.Box.BZ, ROOT.kTRUE)
run.AddModule(Box)
```

PROCESSHITS()

```
Bool_t  Box::ProcessHits(FairVolume* vol)
{
    /** This method is called from the MC stepping */
    //Set parameters at entrance of volume. Reset ELoss.
    if ( gMC->IsTrackEntering() ) {
        fELoss  = 0.;
        fTime   = gMC->TrackTime() * 1.0e09;
        fLength = gMC->TrackLength();
        gMC->TrackPosition(fPos);
        gMC->TrackMomentum(fMom);
    }
    // Sum energy loss for all steps in the active volume
    fELoss += gMC->Edep();

    // Create BoxPoint at exit of active volume
    if ( gMC->IsTrackExiting()    ||
        gMC->IsTrackStop()       ||
        gMC->IsTrackDisappeared()   ) {
        fTrackID  = gMC->GetStack()->GetCurrentTrackNumber();
        fVolumeID = vol->getMCid();
        Int_t detID=0;
        gMC->CurrentVolID(detID);

        if (fVolumeID == detID) {
          return kTRUE; }
        fVolumeID = detID;

        gGeoManager->PrintOverlaps();
```

```
        if (fELoss == 0. ) { return kFALSE; }
        TParticle* p=gMC->GetStack()->GetCurrentTrack();
        Int_t pdgCode = p->GetPdgCode();

        TLorentzVector Pos;
        gMC->TrackPosition(Pos);
        Double_t xmean = (fPos.X()+Pos.X())/2. ;
        Double_t ymean = (fPos.Y()+Pos.Y())/2. ;
        Double_t zmean = (fPos.Z()+Pos.Z())/2. ;


        AddHit(fTrackID,fVolumeID, TVector3(xmean, ymean,  zmean),
               TVector3(fMom.Px(), fMom.Py(), fMom.Pz()), fTime, fLength,
               fELoss, pdgCode);

        // Increment number of muon det points in TParticle
        ShipStack* stack = (ShipStack*) gMC->GetStack();
        stack->AddPoint(ktauBox);
    }

    return kTRUE;
}
```

SAVING THE HITS: ADDHIT()

```
BoxPoint* Box::AddHit(Int_t trackID,Int_t detID,
                      TVector3 pos, TVector3 mom,
                      Double_t time, Double_t length,
                       Double_t eLoss, Int_t pdgCode)
{
    TClonesArray& clref = *fBoxPointCollection;
    Int_t size = clref.GetEntriesFast();
    return new(clref[size]) BoxPoint(trackID,detID, pos, mom,
                                     time, length, eLoss, pdgCode);
}
```

HOW TO MAKE EVERYTHING WORK

THE BOXLINKDEF.H FILE

- In the folder of your detector.
- The ROOTCINT program generates the Streamer(), TBuffer &operator>>() and ShowMembers() methods for ROOT classes as well as the CINT dictionaries needed in order to get access to ones classes via the interpreter
- The LinkDef file tells ROOTCINT for which classes the method interface stubs should be generated.

```
#ifdef __CINT__

#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;

#pragma link C++ class Box+;
#pragma link C++ class BoxPoint+;
#pragma link C++ class BoxContFact+;

#endif
```

The "+" at the end (ACLiC) invokes the dictionary generator and all the rest (essential)

MAKE FAIRSHIP KNOW ABOUT YOUR DETECTOR

- FairShip/CMakeLists.txt
  - To make the FairShip software read the new folder, it is important to insert the title of the folder among those contained in the general CMakeLists.txt file

- shipdata/ShipDetectorList.h
  - In the constructor of the Box class a unique identifier is given to the detector that has to be added to the list of the other identifiers :

```
Box::Box(const char* name, const Double_t BX, const Double_t BY, const Double_t BZ, Bool_t Active,const char* Title)
: FairDetector(name, true, kBox),
```

Box.cxx

ShipDetectorList.h

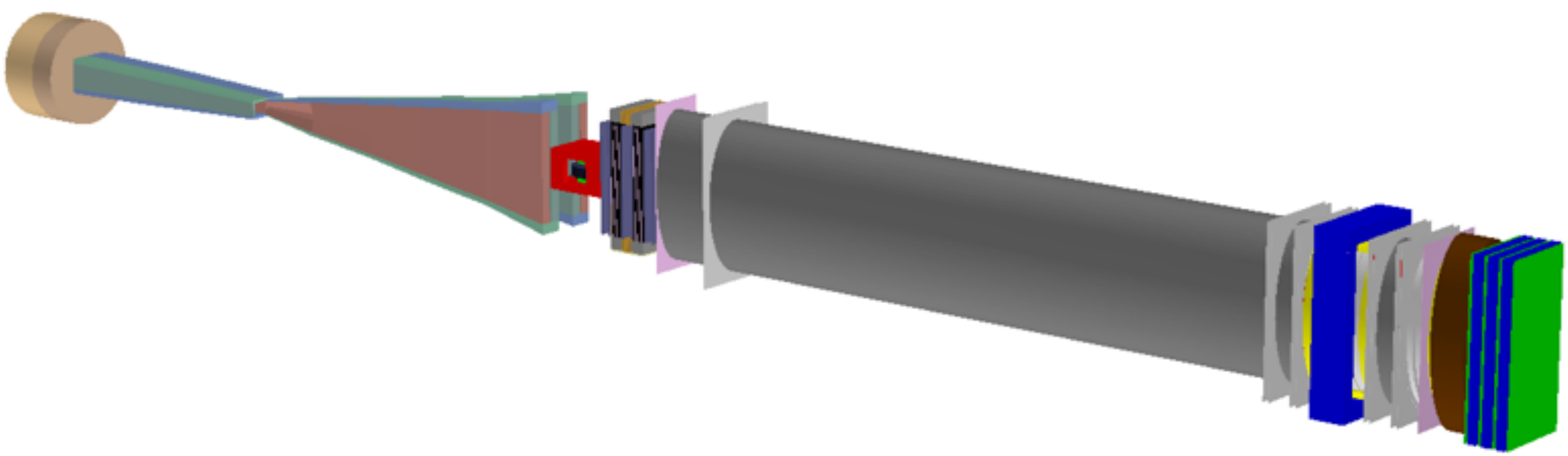
```
#ifndef ShipDetectorList_H
#define ShipDetectorList_H 1

// kSTOPHERE is needed for iteration over the enum. All detectors have to be put before.
enum DetectorId {kVETO, ktauRpc, ktauTarget, kBox, kStraw, kecal, khcal, kMuon ,kTRSTATION};

#endif
```

SUMMARIZING

- To create a new detector folder:
  - Add your folder in the FairShip directory
  - Modify the *FairShip/CMakeLists.txt* adding the name of your folder after ENDIF (NO FAIR ROOT FOUND) with command *add_subdirectory (folder name)*
  - In *Shipdata/ShipDetectorList.h* add the unique identifier you give to your detector (the same you will need to use in one of the constructor of your detector class, look *kBox* in *Box.cxx.*
- In the new folder:
  - Create a *CMakeLists.txt file* and a *xxxLinkDef.h* file (take a look at those in the box folder)
  - If detector is active create the *YourDetectorPoint.h (.cxx)* files (otherwise skip)
  - Create the detector class (*YourDetector.h(.cxx)*) and if the detector is passive do not use functions read hits (see for example *FairShip/passive/ShipMagnet.h*)
    - Check if the media of which your detector is made is already been created in *geometry/media.geo* (otherwise create using info on the slides)
- Add the parameters of your detector in the *geometry/geometry_config.py* file
- Create the geometry object corresponding to your detector by defining it in *python/shipDet_conf.py*

SUMMARIZING

➡This is just a very short introduction on the possibilities given by FairROOT to create new detector geometries.



➡The best way to learn is to try, try and try, also by taking a look at what other people have done.

➡For further information on the geometry package please refer to the FairROOT documentation