SHiP
*Search for Hidden Particles*

# Material and Detector Implementation in FairSHIP

Annarita Buonaura
Università di Napoli & INFN

7th SHiP Collaboration Meeting, CERN, February 10 - 12, 2016

# OUTLINE

- Introduction
- Creating a new detector class
  - The CMakeLists file
  - The Detector Class
    - Describing a detector
    - Creating the shape
    - Creating the medium
    - Creating and positioning the volume
    - Process Hits() , GetCollection() and AddHit() methods
  - The Hits class
- How to make everything work
  - The LinkDef.h file
  - Parameter file
  - Make FairShip know about your detector
- Summary

# Introduction

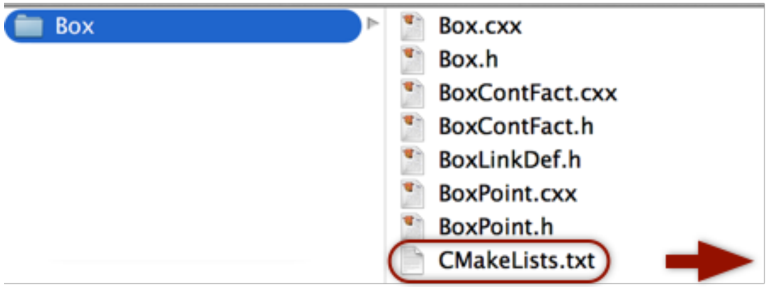
- SHiP geometry environment is mainly based on the ROOT/TGEO package.
- It is a tool for building, browsing, navigating and visualizing detector geometries
- Particle transport is obtained by working in correlation with simulation packages such as GEANT3, GEANT4 and FLUKA

- 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

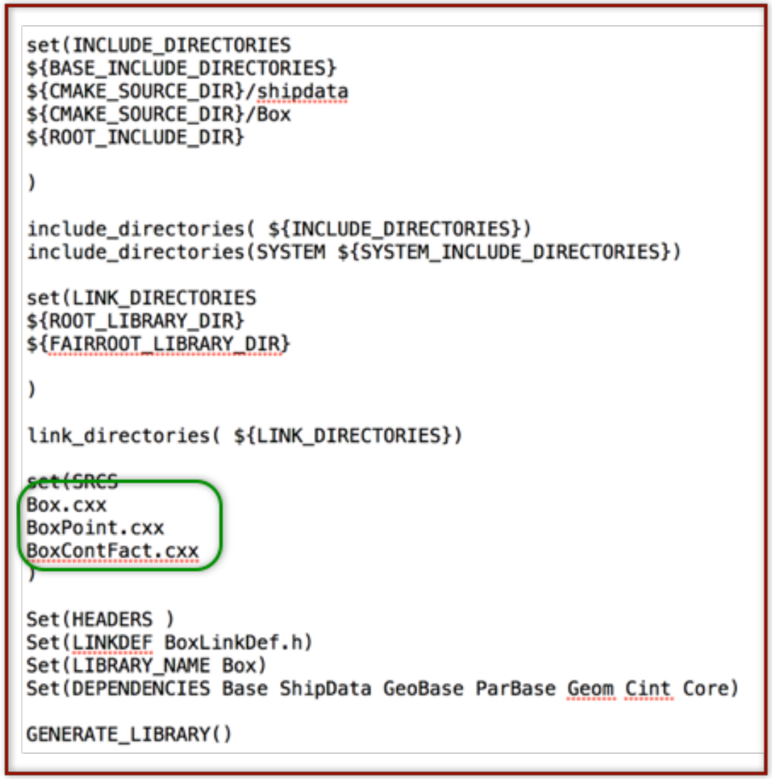
# The CMakeLists File

- 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 .



Box

- Box.cxx
- Box.h
- BoxContFact.cxx
- BoxContFact.h
- BoxLinkDef.h
- BoxPoint.cxx
- BoxPoint.h
- CMakeLists.txt



```
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 Detector Class

- Two files must be created: a .h file and a .cxx usually with the same name

  - The **.h file** (header file) contains declaration of:
    - private/protected member of the class (NB: do not define them here)
    - functions (methods)

  - The **.cxx file** contains:
    - implementation of the methods declared in the .h file

# Describing a detector

- A new detector class can inherit from two different abstract base classes:
    - **FairModule:**
        - Defines a geometry element which does not produce MC points (passive detectors such as /FairSHip/passive/ShipMuonShield.cxx)
    - **FairDetector:**
        - Defines a geometry element with active volumes (e.g. a detector)
        - It is a subclass of the FairModule one which implements extra functions called from the event loop of the MC to make some actions during simulations

# Create a detector volume

- The basic bricks for building-up the model are called **volumes**.
- Volumes are put one inside another making an **in-depth hierarchy**. The one containing all others defines the **“world”** of the model.
- In FairShip the world has already been defined and it can be called using:

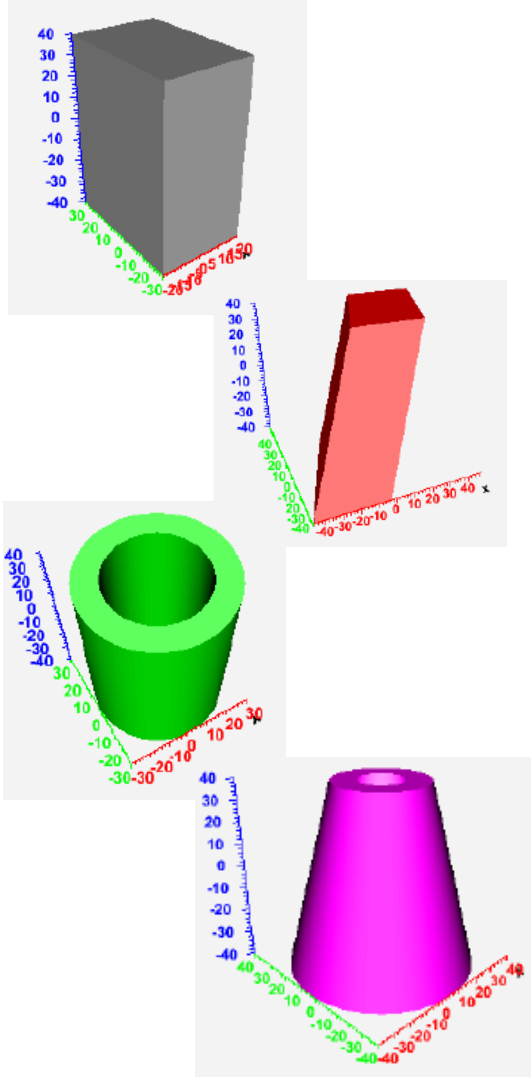
**TGeoVolume** *top = gGeoManager→GetTopVolume();

- To define a volume it is necessary to create **media** and **shapes**.
- Both containers and contained volumes must be created before linking them together:  a relative transformation matrix must also be provided.
- Any volume has to be positioned somewhere otherwise it will not be considered as part of the geometry.

# Create a detector shape

- Each volume has a **shape**.
- It provides the definition of the local coordinate system of the volume.
- Any shape has to derive from the base **TGeoShape** class.
- 20 basic (***primitive***) shapes are already provided:
    - Boxes: TGeoBBox class
    - Parallelepipid: TGeoPara class
    - Trapezoids: TGeoTrd1 class
    - Cones – TGeoCone Class
    - Arbitrary 8 vertices shapes - TGeoArb8 class
    - Tubes – TGeoTube Class

  …
- Composite shapes can also be created as a result of Boolean operations between primitives



# Create a detector shape (2)

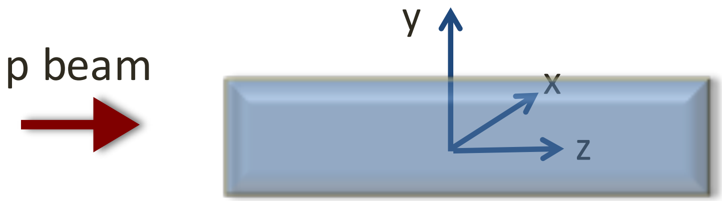
- All primitives have constructors like:

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

- Example:

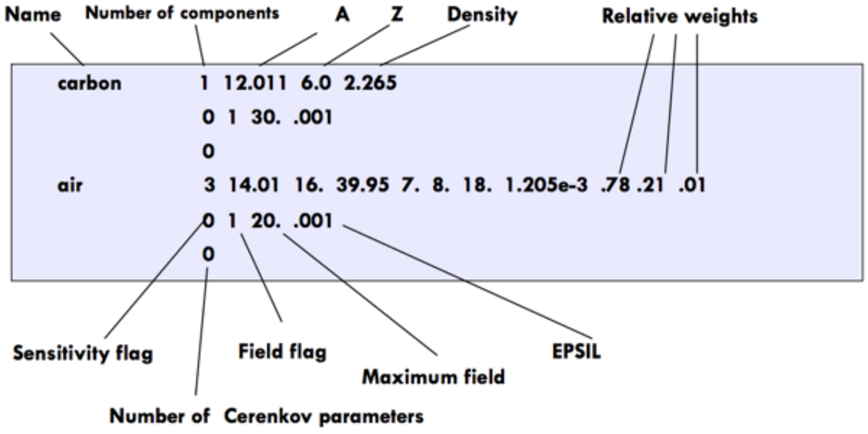
| | | |
|---|---|---|
| `TGeoBBox(Double_t  fX,` | ⟵ | Half length in X |
| `Double_t  fY,` | ⟵ | Half length in Y |
| `Double_t  fZ,` | ⟵ | Half length in Z |
| `Double_t  *origin=0);` | ⟵ | Origin of the box: (0,0,0) by default |

NB: The default length units are centimetres



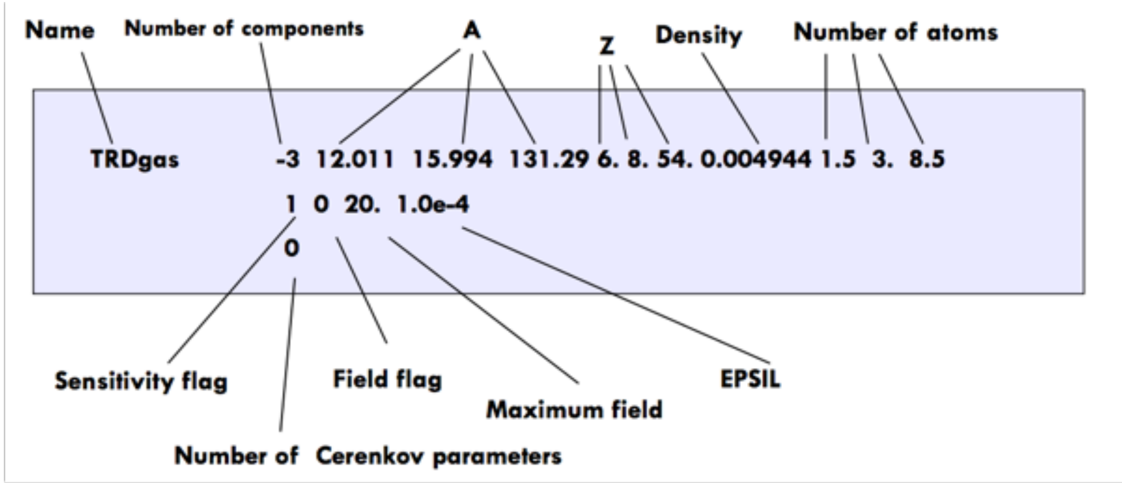
# Defining the media

- 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 ....)
- 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

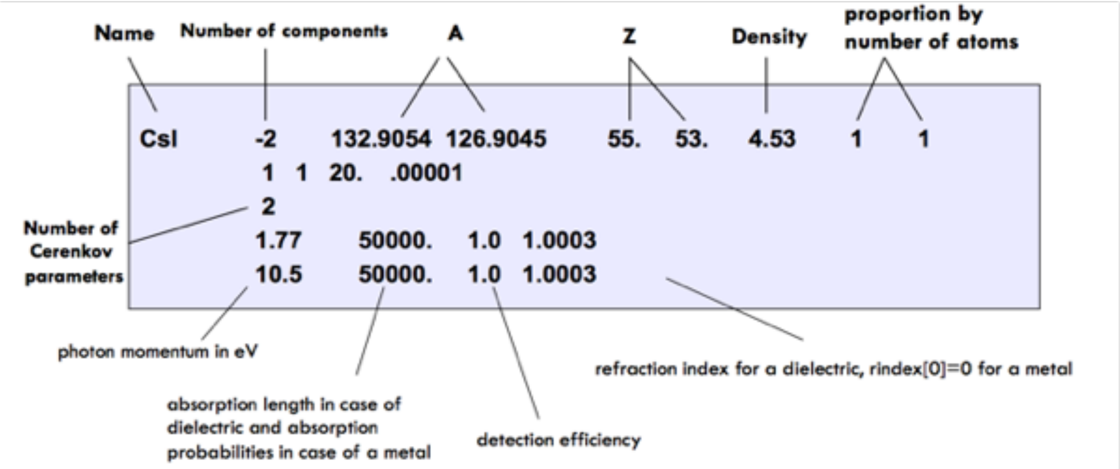


```
carbon        1  12.011  6.0  2.265
              0  1  30.  .001
              0
air           3  14.01  16.  39.95  7.  8.  18.  1.205e-3  .78 .21  .01
              0  1  20.  .001
              0
```

# Defining the media (2)



```
TRDgas      -3  12.011  15.994  131.29 6. 8. 54. 0.004944 1.5  3.  8.5
                 1  0  20.  1.0e-4
                 0
```

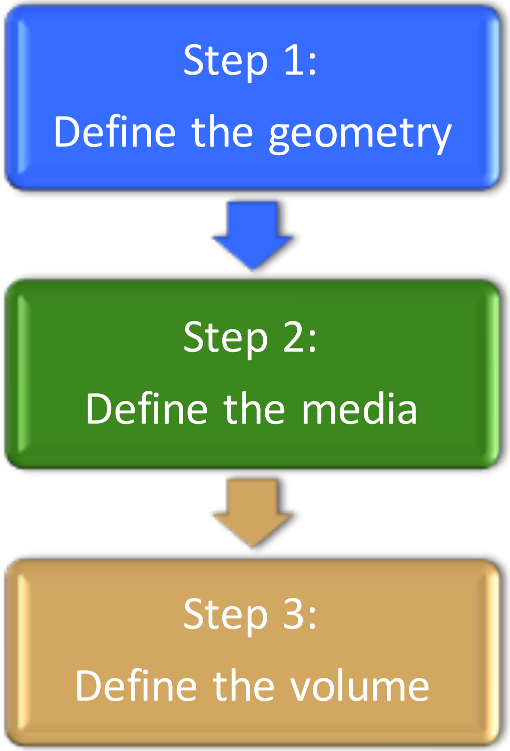


```
CsI        -2      132.9054  126.9045      55.    53.    4.53     1       1
            1   1   20.    .00001
            2
           1.77       50000.    1.0   1.0003
           10.5       50000.    1.0   1.0003
```

Name
Number of components
A
Z
Density
proportion by number of atoms
Number of Cerenkov parameters
photon momentum in eV
absorption length in case of dielectric and absorption probabilities in case of a metal
detection efficiency
refraction index for a dielectric, rindex[0]=0 for a metal

# Create a detector volume (2)

- Basic strategy



Step 1:
Define the geometry

Step 2:
Define the media

Step 3:
Define the volume

```
TGeoBBox *fBox = new TGeoBBox(Double_t fX,
Double_t fY, Double_t fZ);
```

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

```
TGeoVolume *fBoxVol = new TGeoVolume("volBox", fBox, Fe);
```

# Positioning the volume

- Before positioning a volume, its mother volume must be created
- Daughter volumes must not extrude their mother shape.
- Positions of daughter volumes with respect to the center of mother volume must be known: a geometrical transformation when positioning daughter volumes must be provided.
- Volumes in the same container must not overlap with each other
- If the detector consists of a repetition of unitary cells (e.g. 10 iron layers):
    - *Not* create a different shape and  a different volume for each cell
    - It is enough to replicate the ones that have been already created

# Positioning the volume (2)

## Example: Box positioned in top volume

```
TGeoVolume *top = gGeoManager→GetTopVolume();
```

Call top volume

```
TGeoBBox *fBox = new TGeoBBox(Double_t fX, Double_t fY, Double_t fZ);
InitMedium("iron");
TGeoMedium *Fe = gGeoManager->GetMedium(“iron");
TGeoVolume *fBoxVol  = new TGeoVolume("volBox", fBox, Fe);
TGeoTranslation *fT = new TGeoTranslation(fTx, fTy, fTz);
top -> AddNode(fBoxVol, 1, fT);
```

Define Box

Define Box material

Define Box volume

Position of the box w.r.t. top volume

Position the box in top volume

Mother volume

Number of replica

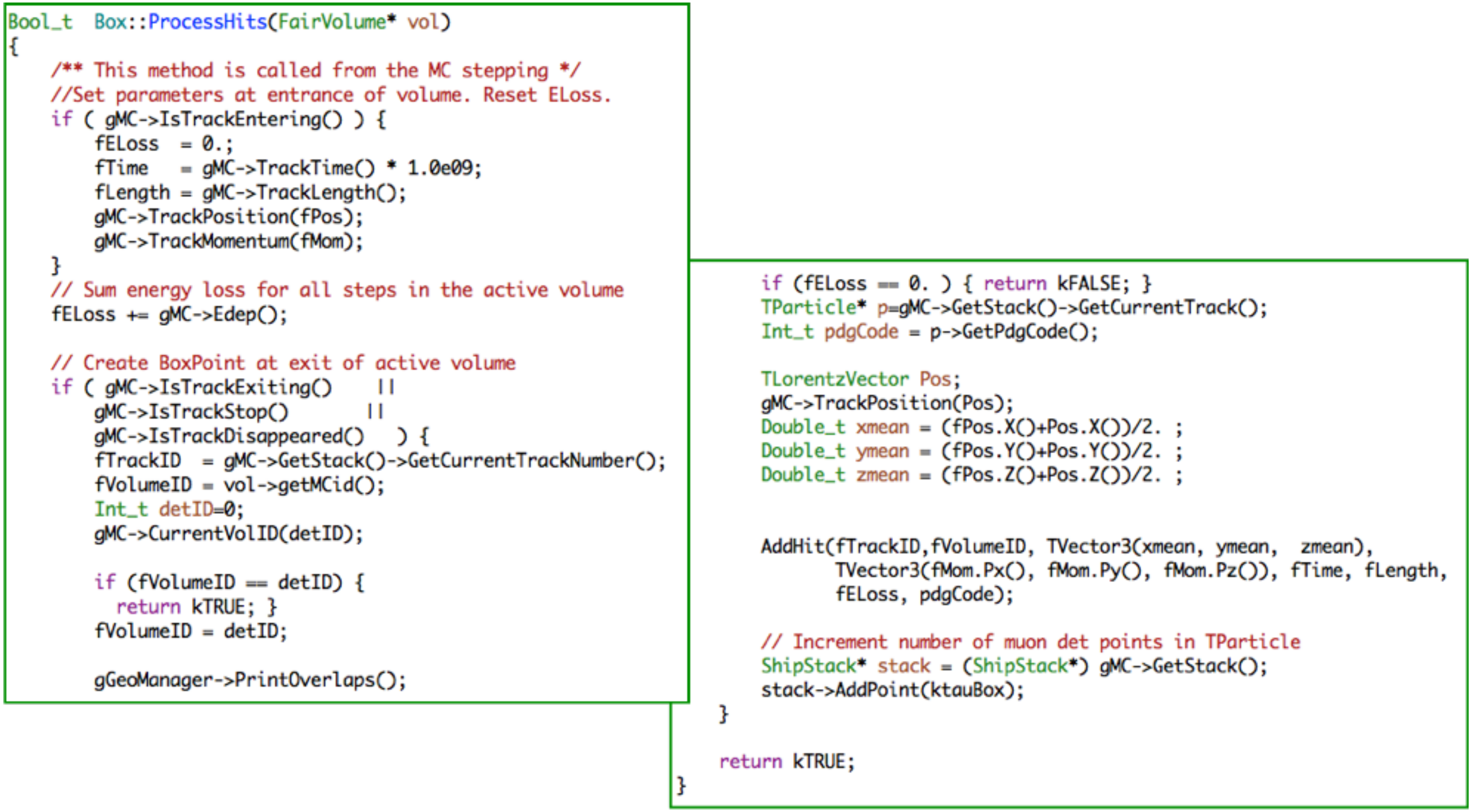
TGeoTranslation

```
TGeoBBox *fScint = new TGeoBBox(Double_t fX, Double_t fY, Double_t fScintZ);
InitMedium(“scint”);
TGeoMedium *Fe = gGeoManager->GetMedium(“Scintillator");
TGeoVolume *fScintVol  = new TGeoVolume("volScint", fScint, scint);
for(Int_t n =0; n< nReplica; n++){
        TGeoTranslation *t = new TGeoTranslation(0,0,-fZ+n*(fScintZ+0.1)]fScint/2);
        fBox -> AddNode(VBox, n, t);
}
```

Translations along x,y,z w.r.t. center of mother volume

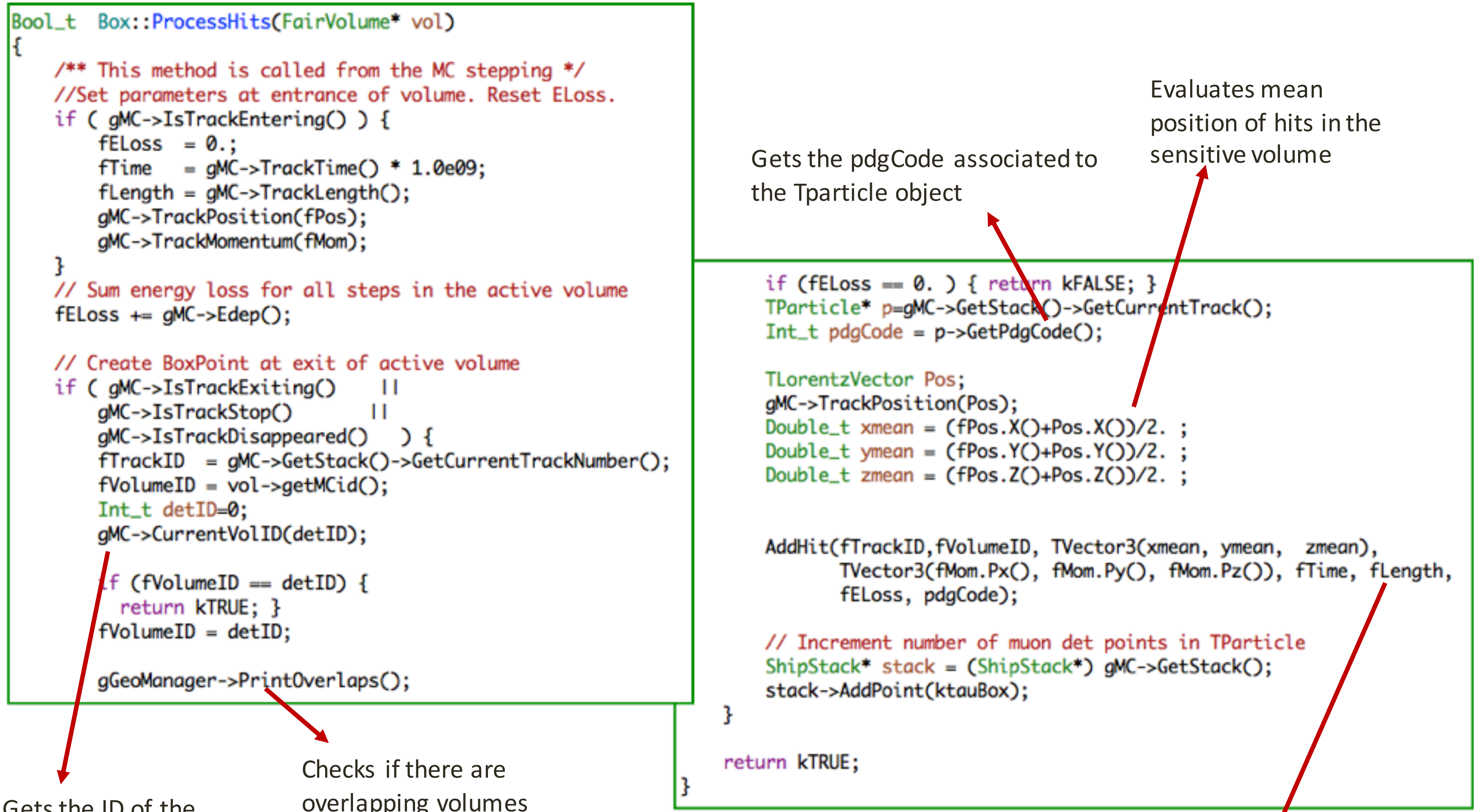
Number of replica

# 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;
}
```

# 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;
}
```

Gets the pdgCode associated to the Tparticle object

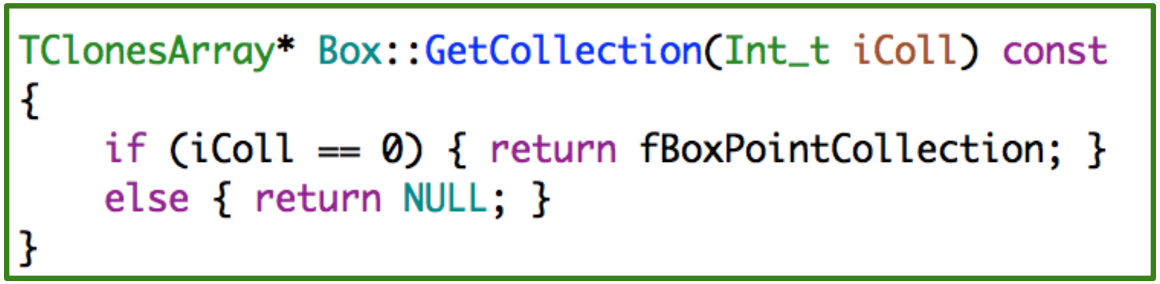
Evaluates mean position of hits in the sensitive volume

Gets the ID of the volume where the hit was released

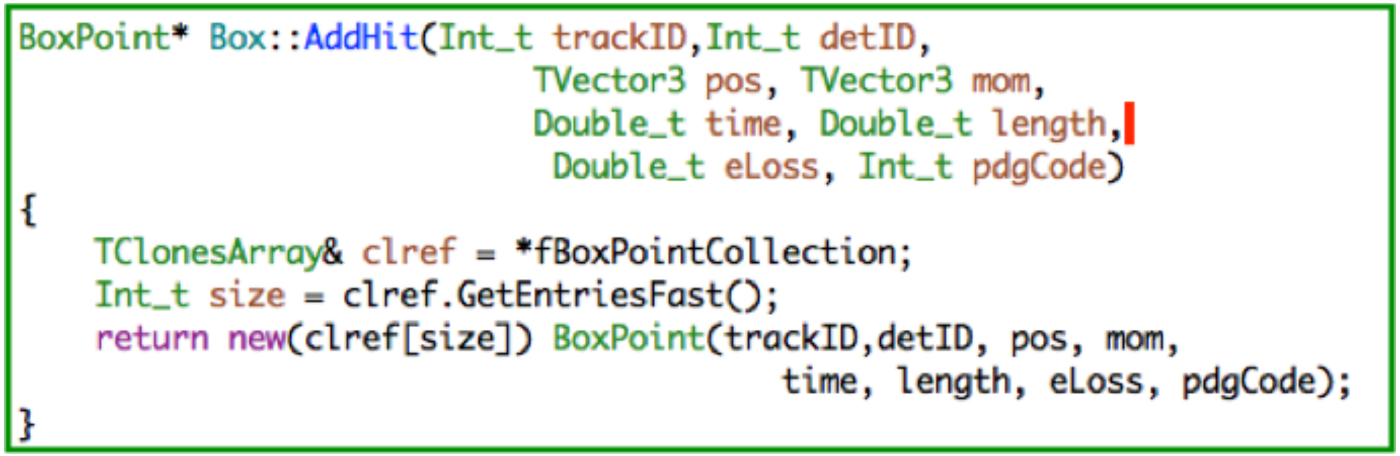
Checks if there are overlapping volumes

Adding hits to save

# Saving the hits: GetCollection() & AddHit()



```
TClonesArray* Box::GetCollection(Int_t iColl) const
{
    if (iColl == 0) { return fBoxPointCollection; }
    else { return NULL; }
}
```



```
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);
}
```

# The hits class

- Usually created with a name ***Point
- It inherits from the FairMCPoint class

- As for the detector class:
    - .h file:
        - Point constructor with/without arguments
            - **trackID** = Index of MC track
            - **detID** = Detector ID
            - **pos** = Coordinates at the center of the active volume [cm]
            - **mom** = Momentum of track at entrance [GeV]
            - **tof** = Time since event start [ns]
            - **length** = Track length since creation [cm]
            - **eLoss** = Energy deposit [GeV]
            - **pdgcode** = Pdg Code of the track
        - Definition of functions acting on the class
            - Example: **Int_t** PdgCode() **const** {**return** fPdgCode;}
    - .cxx file:
        - … of fuctions defined in .h file

# HOW TO MAKE EVERYTHING WORK …

# Parameter File

- 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;
```

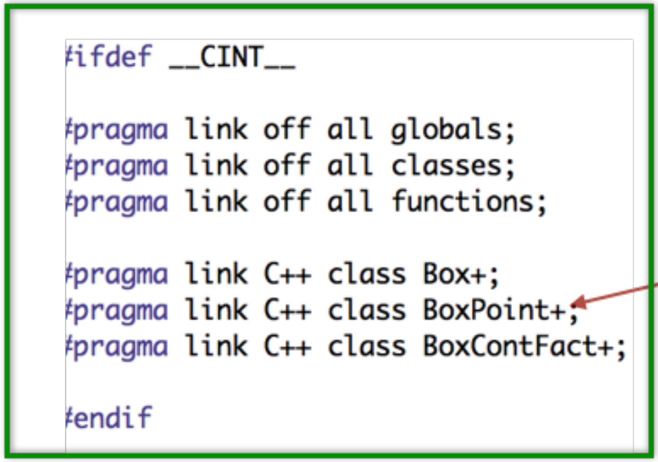
- Geometry objects are created by python/shipDet_conf.py and declared to the run manager FairRunSim()

Example

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

# The LinkDef.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.



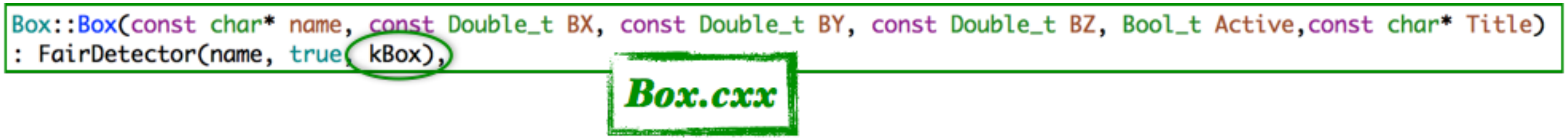
```
#ifdef __CINT__

#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

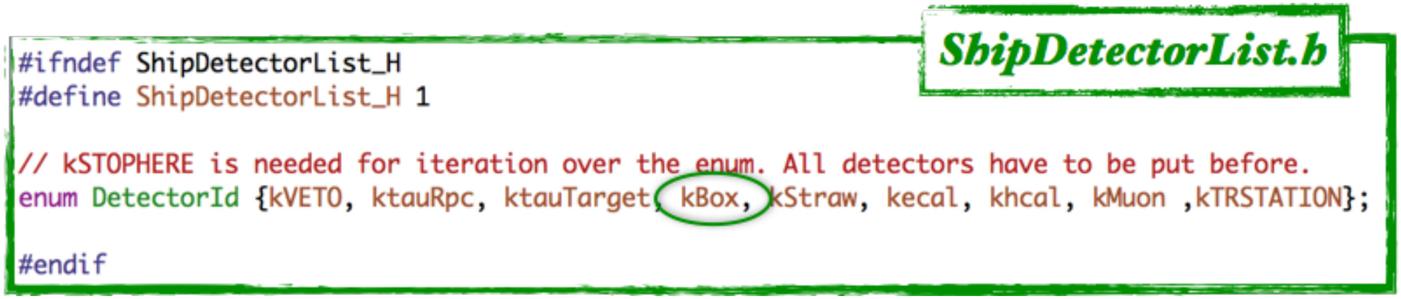
- ***FairShip/CMakeLists.txt***
- To make the FairShip software read the new folder, it is important to insert the title of the folder among those contained in the general CMakeLists.txt file

- ***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**

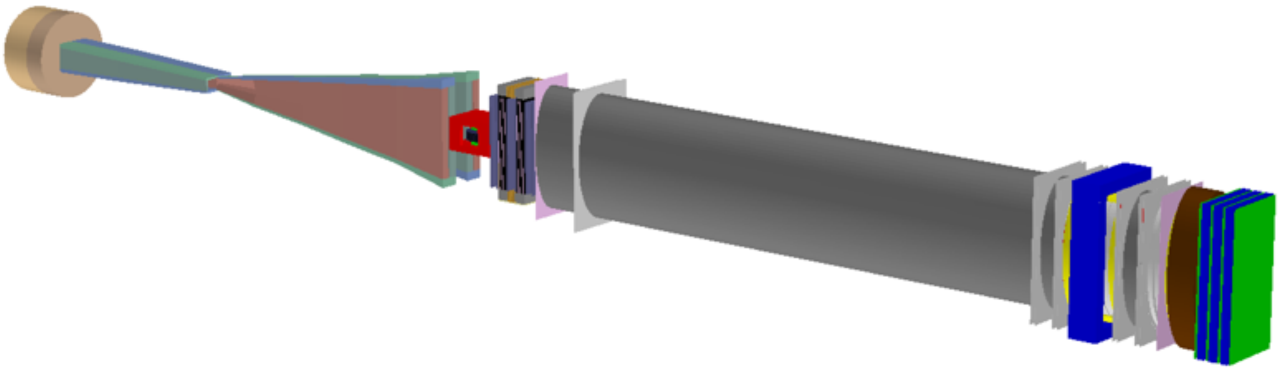
```
#ifndef ShipDetectorList_H
#define ShipDetectorList_H 1

// 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.
- In the new folder:
  - Create a *CMakeLists.txt* file and a ****LinkDef.h* file
  - If detector is active create the ****Point.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

# BACK – UP SLIDES

# InitMedium Function

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);
}
```

# Magnetic Field

- The value of the magnetic field can be defined as a private member of the detector class.

**Example:**

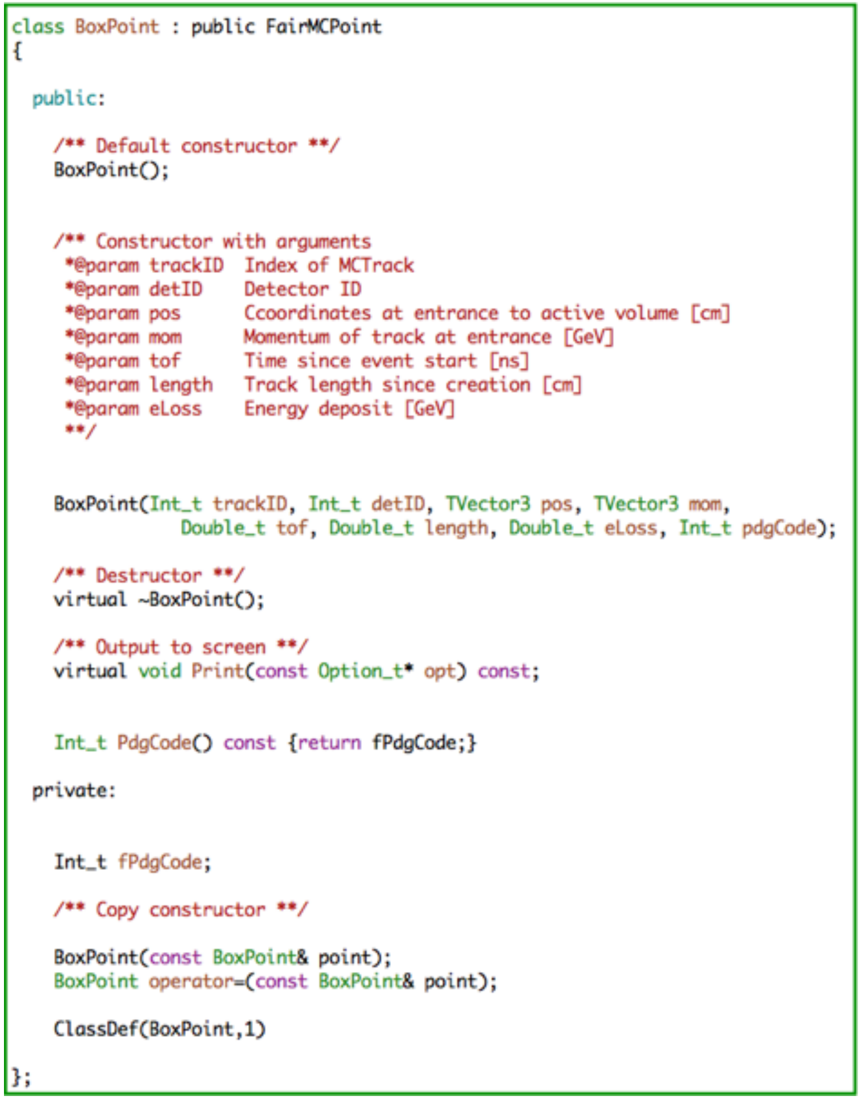
```
TGeoUniformMagField *magField = new TGeoUniformMagField(0.,-MagneticField,0.);
fBoxVol->SetField(magField);
```

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

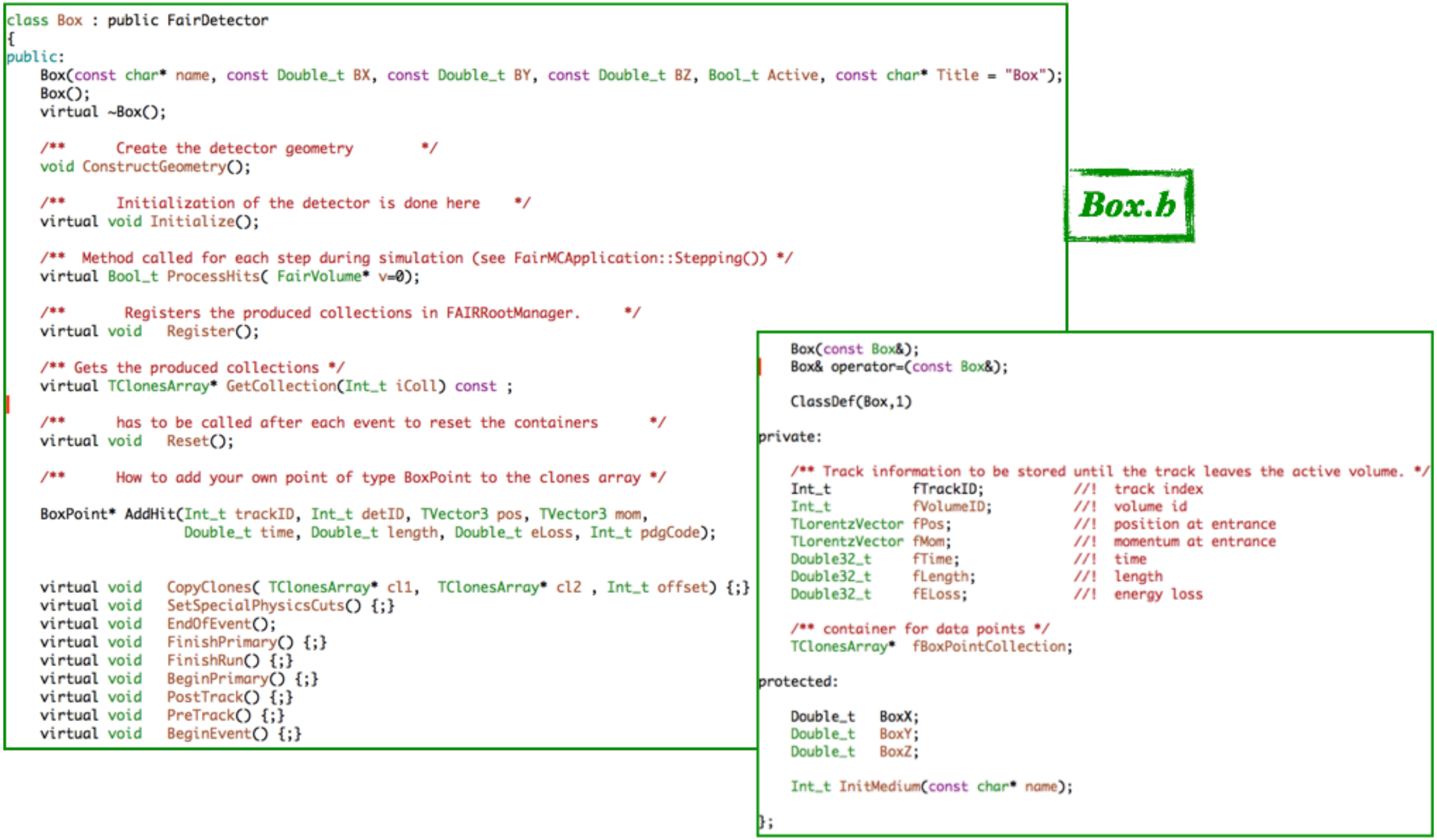
```
#manipulate G4 geometry to enable magnetic field in active shielding, VMC can't do it.
import geomGeant4
geomGeant4.setMagnetField() # ('dump') for printout of mag fields
```

# The Hits class (example)



```
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)

};
```

# The Detector Class (2)



Box.h

```
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(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);

};
```