```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ********************************************************************
//
/// \file runAndEvent/RE02/src/DetectorConstruction.cc
/// \brief Implementation of the DetectorConstruction class
//
//
// $Id: DetectorConstruction.cc 75682 2013-11-05 09:11:19Z gcosmo $
//

#include "DetectorConstruction.hh"
#include "FluenceEnergyDistributionSD.hh"

#include "G4PSEnergyDeposit3D.hh"
#include "G4PSDoseDeposit3D.hh"
#include "G4PSCellFlux3D.hh"

#include "G4SDParticleWithEnergyFilter.hh"
#include "G4SDParticleFilter.hh"
#include "G4SDChargedFilter.hh"

#include "G4NistManager.hh"
#include "G4Material.hh"
#include "G4Box.hh"
#include "G4Orb.hh"
#include "G4Tubs.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
#include "G4SDManager.hh"
#include "G4Region.hh"

#include "G4PVParameterised.hh"
#include "NestedPhantomParameterisation.hh"

#include "G4VisAttributes.hh"
#include "G4Colour.hh"

#include "G4SystemOfUnits.hh"
#include "G4PhysicalConstants.hh"
#include "G4ios.hh"

//============================================================================
```

```
DetectorConstruction::DetectorConstruction()
 : G4VUserDetectorConstruction(), fCheckOverlaps(0)
{
    fNx = fNy = fNz = 1; // Number of segmentation of water phantom.

        //------------------------------------------------------------------------
        // Reading energy binning for Spectral Values

        std::ifstream readBinning;
        readBinning.open("inputs/Binning.ebin");
        if(!readBinning)
        {
            G4cout << "Cannot find or open Binning.ebin" << G4endl;
            exit(-1);
        }

        G4bool doReading = true;
        while(doReading)
        {
            G4double LeftEnergyBinEdge;
            readBinning >> LeftEnergyBinEdge;
            if(readBinning.eof()) break;
            fEnergyBinning.push_back(LeftEnergyBinEdge*MeV);
            G4cout << "Binning Values: " << LeftEnergyBinEdge << G4endl;
        }
        readBinning.close();


        fUSDParticles.push_back("all");
        fUSDParticles.push_back("proton");
        fUSDParticles.push_back("neutron");
//      fUSDParticles.push_back("gamma");
//      fUSDParticles.push_back("e-");
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
DetectorConstruction::~DetectorConstruction()
{;}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
 G4VPhysicalVolume* DetectorConstruction::Construct()
{
#ifdef DEFMATERIAL
        G4NistManager* NISTman = G4NistManager::Instance();

        fAir    = NISTman->FindOrBuildMaterial("G4_AIR");
        fPMMA   = NISTman->FindOrBuildMaterial("G4_PLEXIGLASS");

        fthsc_H = new G4Element("TS_H_of_Water","h_Water",1.0,1.0079*g/mole);
        fO = NISTman->FindOrBuildElement("O");

        fWater = new G4Material("G4_WATER",1.000*g/cm3,2,kStateLiquid,295*kelvin);
        fWater->SetChemicalFormula("H_2O");
        fWater->AddElement(fthsc_H,2);
        fWater->AddElement(fO,1);

        if(!fAir || !fWater || !fPMMA || !fthsc_H || !fO )
        {
                G4cerr << "Material was not found or build!" << G4endl;
                exit(-1);
        }
#else
        fthsc_H = new G4Element("TS_H_of_Water","h_water",1.0,1.0079*g/mole);
```

```
        G4Isotope* O16 = new G4Isotope("O16",8,16,15.995*g/mole);
        fO = new G4Element("Oxygen","O",1);
        fO->AddIsotope(O16,1.);

        G4Isotope* H1 = new G4Isotope("H1",1,1,1.0078*g/mole);
        G4Element* H = new G4Element("Hydrogen","H",1);
        H->AddIsotope(H1,1.);

        G4Isotope* C12 = new G4Isotope("C12",6,12,12.*g/mole);
        G4Element* C = new G4Element("Carbon","C",1);
        C->AddIsotope(C12,1.);

        G4Isotope* N14 = new G4Isotope("N14",7,14,14.007*g/mole);
        G4Element* N = new G4Element("Nitrogen","N",1);
        N->AddIsotope(N14,1.);

        G4Isotope* Ar18 = new G4Isotope("Ar18",18,40,39.948*g/mole);
        G4Element* Ar = new G4Element("Argon","Ar",1);
        Ar->AddIsotope(Ar18,1.);

        if(!O16 || !fO || !H1 || !H || !C12 || !C || !N14 || !N || !Ar18 || !Ar )
        {
                G4cerr << "Problem with Isotope Materials!" << G4endl;
                exit(-1);
        }


        fAir = new G4Material("G4_AIR",0.001205*g/cm3,4,kStateGas,293.15*kelvin);
        fAir->AddElement(H,0.012827);
        fAir->AddElement(C,0.000124);
        fAir->AddElement(N,0.755268);
        fAir->AddElement(Ar,0.231781);

        fWater = new G4Material("G4_WATER",0.998*g/cm3,2,
                                                kStateLiquid,293.15*kelvin);
        fWater->SetChemicalFormula("H_2O");
        fWater->AddElement(fthsc_H,2);
        fWater->AddElement(fO,1);
        fWater->GetIonisation()->SetMeanExcitationEnergy(78.0*eV);

        fPMMA = new G4Material("PMMA",1.18*g/cm3,3,kStateSolid,293.15*kelvin);
        fPMMA->AddElement(H,0.080538158);
        fPMMA->AddElement(C,0.599848);
        fPMMA->AddElement(fO,0.319618);


        if(!fAir || !fWater || !fPMMA || !fthsc_H || !fO )
        {
                G4cerr << "Material was not found or build!" << G4endl;
                exit(-1);
        }

#endif

        G4cout << G4endl << "The materials defined are : " << G4endl << G4endl;
        G4cout << *(G4Material::GetMaterialTable()) << G4endl;



//=====================================================================
//      Definitions of Solids, Logical Volumes, Physical Volumes
//=====================================================================

        //-------------
```

```
        // World Volume
        //-------------

        G4ThreeVector worldSize = G4ThreeVector(250.*cm, 250.*cm, 250.*cm);

        G4Box * solidWorld
                                = new G4Box("world", worldSize.x()/2.,
                                                                    worldSize.y()/
2.,
                                                                    worldSize.z()/
2.);
        G4LogicalVolume * logicWorld
                                = new G4LogicalVolume(solidWorld, fAir, "World", 0, 0,
0);

        G4VPhysicalVolume * physiWorld
                                = new G4PVPlacement(0,              // no rotation
                                                    G4ThreeVector(), // at
(0,0,0)
                                                                    logicWorld,
// its logical volume
                                                                    "World",
// its name
                                                                    0,
// its mother  volume
                                                                    false,
// no boolean operations
                                                                    0);
// copy number

//--------------------------------------------------------------------------


    G4double temperature = 293.15*kelvin;

    G4NistManager* NistManager = G4NistManager::Instance();

    G4Element* Al = NistManager->FindOrBuildElement("Al");
    G4Element* O  = NistManager->FindOrBuildElement("O");
    G4Element* C  = NistManager->FindOrBuildElement("C");
    G4Element* H  = NistManager->FindOrBuildElement("H");
    G4Element* Si = NistManager->FindOrBuildElement("Si");
    G4Element* B  = NistManager->FindOrBuildElement("B");
    G4Element* N  = NistManager->FindOrBuildElement("N");
    G4Element* F  = NistManager->FindOrBuildElement("F");
    G4Element* Fe = NistManager->FindOrBuildElement("Fe");
    G4Element* Pb = NistManager->FindOrBuildElement("Pb");
    G4Element* Cd = NistManager->FindOrBuildElement("Cd");

        //------------------------------------------------------------------------

    G4Isotope* Li6 = new G4Isotope("Li6",3,6,6.0151228*g/mole);
    G4Isotope* Li7 = new G4Isotope("Li7",3,7,7.0160045*g/mole);
    G4Element* Li  = new G4Element("enriched_Li","Li",2);
    Li->AddIsotope(Li6,0.96);
    Li->AddIsotope(Li7,0.04);

        //------------------------------------------------------------------------

    // Thermal scattering hydrogen:
    G4Element* thsc_H = new G4Element("TS_H_of_Polyethylene","H_POLYETHYLENE",
                                                                    1.0,1.0079*g/
mole);
```

```
        //------------------------------------------------------------------------

    Air = NistManager->FindOrBuildMaterial("G4_AIR");
    Vacuum = NistManager->FindOrBuildMaterial("G4_Galactic");
    G4Material* Ceramic = NistManager->FindOrBuildMaterial("G4_ALUMINUM_OXIDE");
    G4Material* Epoxidharz = new G4Material("Epoxidharz",1.2*g/cm3,4,

kStateSolid,temperature);
    Epoxidharz->AddElement(C,0.7545);
    Epoxidharz->AddElement(H,0.0715);
    Epoxidharz->AddElement(N,0.0065);
    Epoxidharz->AddElement(O,0.1675);

        //------------------------------------------------------------------------

        G4Material* Lead = new G4Material("Blei",11.342*g/cm3,1,

kStateSolid,temperature);
    Lead->AddElement(Pb,1);

        //------------------------------------------------------------------------

        G4Material* Iron = new G4Material("Stahl",7.874*g/cm3,1,

kStateSolid,temperature);
    Iron->AddElement(Fe,1);

        //------------------------------------------------------------------------

        G4Material* SiliconOxide = new G4Material("Siliziumdioxid",2.19*g/cm3
                                                  /* bis 2.66*g/cm3*/,
2,kStateSolid,temperature);
    SiliconOxide->SetChemicalFormula("SiO_2");
    SiliconOxide->AddElement(Si,(G4int)1);
    SiliconOxide->AddElement(O,(G4int)2);

        //------------------------------------------------------------------------

        G4Material* Silicon = new G4Material("Silizium",2.336*g/cm3,1,

kStateSolid,temperature);
    Silicon->AddElement(Si,1);

        //------------------------------------------------------------------------

    G4Material* Aluminium = new G4Material("Aluminium",2.7*g/cm3,1,

kStateSolid,temperature);
    Aluminium->AddElement(Al,1);

        //------------------------------------------------------------------------

    G4Material* Cadmium = new G4Material("Cadmium",8.65*g/cm3,1,

kStateSolid,temperature);
    Cadmium->AddElement(Cd,1);

        //------------------------------------------------------------------------

    G4Material* Wax = new G4Material("Polyethylenwax",/*0.94*g/cm3 bis
                                                       0.97*g/cm3*/
0.98*g/cm3/*gemessen*/,2,

kStateSolid,temperature);
```

```
    Wax->SetChemicalFormula("(C_2H_4)-Polyethylene");
    Wax->AddElement(C,(G4int)1);
    Wax->AddElement(thsc_H,(G4int)2); // thermal scattering auch für PE

        //------------------------------------------------------------------------

        G4Material* B4C = new G4Material("Borcarbid",1.32*g/cm3/*Packung*/,

2,kStateSolid,temperature);
    // B4C kein Eintrag in "Chemical Formula" vorhanden...
    B4C->AddElement(B,(G4int)4);
    B4C->AddElement(C,(G4int)1);

        //------------------------------------------------------------------------

        G4Material* BoronCarbidEpoxid = new G4Material("B4C_Epoxidharz",
                                                                    1.585*g/cm3/
*gemessen*/,2,

kStateSolid,temperature);
    BoronCarbidEpoxid->AddMaterial(B4C,0.6);
    BoronCarbidEpoxid->AddMaterial(Epoxidharz,0.4);

        //------------------------------------------------------------------------

    G4Material* Glue = new G4Material("Glue",0.83*g/cm3/*???*/,2,

kStateLiquid,temperature);
    Glue->AddElement(C,(G4int)1);
    Glue->AddElement(H,(G4int)2);

        //------------------------------------------------------------------------

    G4Material* LiF = new G4Material("LiF",2.55*g/cm3/*berechnet*/,2,

kStateSolid,temperature);
    LiF->SetChemicalFormula("LiF");
    LiF->AddElement(Li,(G4int)1);
    LiF->AddElement(F,(G4int)1);

        //------------------------------------------------------------------------

    G4Material* LiF_CH2 = new G4Material("LiF_CH2",2.46*g/cm3/*berechnet*/,2,

kStateSolid,temperature);
    LiF_CH2->AddMaterial(LiF,0.95);
    LiF_CH2->AddMaterial(Glue,0.05);

        //------------------------------------------------------------------------

//    G4Material* h2o = NistManager->FindOrBuildMaterial("G4_WATER");
        //  G4Material* pmma = NistManager->FindOrBuildMaterial("G4_PLEXIGLASS");
    // Thermal scattering PMMA:

        //------------------------------------------------------------------------



    G4Material* pmma = new G4Material("PMMA",1.19*g/cm3,3,kStateSolid,
                                                                    temperature);
    pmma->SetChemicalFormula("(C_5H_8O-2)-Polymethil_Methacrylate");
    pmma->AddElement(thsc_H,(G4int)8);
    pmma->AddElement(C,(G4int)5);
    pmma->AddElement(O,(G4int)2);
```

```
//Phantom shape, logical volume and position
        G4ThreeVector  PMMAPhantomSize = G4ThreeVector(30.*cm,30.*cm,30.*cm);
        G4ThreeVector  PMMAPhantomPos  = G4ThreeVector(0.,0.,0.);
        G4RotationMatrix* PMMAPhantomRot = new G4RotationMatrix();

        fSolidPMMAPhantom = new G4Box("solidPMMAPhantom",
                                                              PMMAPhantomSize.x()/2.,
                                                              PMMAPhantomSize.y()/2.,
                                                              PMMAPhantomSize.z()/2.);

        fLogicPMMAPhantom = new G4LogicalVolume(fSolidPMMAPhantom,pmma,

"logicPMMAPhantom",0,0,0);

        fPhysiPMMAPhantom = new G4PVPlacement(PMMAPhantomRot,

PMMAPhantomPos,

fLogicPMMAPhantom,

"physiPMMAPhantom",

logicWorld,
                                                                    false,
                                                                    0,
                                                                    true);


    // Gehäuse, Füllung und Plazierung:

    G4double detectorThickness = 400*um;
    G4double numberOfLayers        = 100;
    G4double layerThickness    = detectorThickness / (double)numberOfLayers;
    G4double sensorWidth           = sqrt(200)*mm;
    G4double sensorWidthExt    = 16.15*mm;

    // Fast:
    G4double fast_leadThicknessFront = 1.0*mm;
    G4double fast_waxThickness           = 2.5*mm;
    G4double fast_ceramicThickness   = 0.5*mm;
    G4double fast_leadThicknessBack  = 1.0*mm;

    G4double fast_sensorThickness = fast_leadThicknessBack
                                                                + fast_ceramicThickness
                                                                + detectorThickness
                                                                + fast_waxThickness
                                                                +
fast_leadThicknessFront;

        //========================================================================
// Fast Sensor housing shape, logical volume, position and rotation matrix before that
    G4Box* fast_housing_s = new G4Box("fast_housing_s",

sensorWidthExt/2.,

sensorWidthExt/2.,

fast_sensorThickness/2.);
        G4LogicalVolume*
        fast_housing_log = new G4LogicalVolume(fast_housing_s,Lead,

"fast_housing_log");
```

```
//      fast_housing_log->SetVisAttributes(lead_vis);
        G4RotationMatrix* rotFast = new G4RotationMatrix();
        rotFast->rotateY(180.*deg);


        G4PVPlacement* fast_housing =
                                                new G4PVPlacement(rotFast,
#ifdef FAST_DET
                                                                  //
G4ThreeVector(0.,0.,PMMAPhantomSize.x()/2.+fast_sensorThickness/2.-5.*cm),
                                                                  G4ThreeVector(0.,10.*cm,
0.),
#else
                                                                  //G4ThreeVector(12.*cm,
12.*cm,-15.*cm),
                                                                  G4ThreeVector(0.,10.*cm,
0.),
#endif
                                                                  "Fast_Housing",
                                                                  fast_housing_log,
                                                                  fPhysiPMMAPhantom,
                                                                  false,
                                                                  0,
                                                                  1);

        //------------------------------------------------------------------------

    G4Box* fast_leadFront_s = new G4Box("fast_leadFront_s",

sensorWidth/2.,

sensorWidth/2.,

fast_leadThicknessFront/2.);

    G4Box* fast_wax_s = new G4Box("fast_wax_s",
                                                                    sensorWidth/2.,
                                                                    sensorWidth/2.,
                                                                    fast_waxThickness/2.);

    G4Box* fast_ceramic_s = new G4Box("fast_ceramic_s",
                                                                            sensorWidth/2.,
                                                                            sensorWidth/2.,

fast_ceramicThickness/2.);

    G4Box* fast_leadBack_s = new G4Box("fast_leadBack_s",
                                                                             sensorWidth/
2.,
                                                                             sensorWidth/
2.,

fast_leadThicknessBack/2.);

    G4Box* solidFastSens = new G4Box("solidFastSens",
                                                                             sensorWidth/
2.,
                                                                             sensorWidth/
2.,

detectorThickness/2.);
        //------------------------------------------------------------------------

    G4LogicalVolume* fast_leadFront_log =
```

```
                                                    new
G4LogicalVolume(fast_leadFront_s,

Lead,

"fast_leadFront_log");
//    fast_leadFront_log->SetVisAttributes(lead_vis);

        //------------------------------------------------------------------------

        G4LogicalVolume* fast_wax_log = new G4LogicalVolume(fast_wax_s,

        Wax,

        "fast_wax_log");
//    fast_wax_log->SetVisAttributes(converter_vis);

        //------------------------------------------------------------------------

    G4LogicalVolume* fast_ceramic_log = new G4LogicalVolume(fast_ceramic_s,

        Ceramic,

        "fast_ceramic_log");
 //   fast_ceramic_log->SetVisAttributes(ceramic_vis);

        //------------------------------------------------------------------------

    G4LogicalVolume* fast_leadBack_log = new G4LogicalVolume(fast_leadBack_s,

        Lead,

        "fast_leadBack_log");
 //   fast_leadBack_log->SetVisAttributes(lead_vis);

        //------------------------------------------------------------------------

    G4LogicalVolume* logicFastSens = new G4LogicalVolume(solidFastSens,

         Silicon,

         "logicFastSens");


        G4Region* detectorRegion = new G4Region("detectorRegion");
    detectorRegion->AddRootLogicalVolume(fast_housing_log);
//      detectorRegion->AddRootLogicalVolume(albedo_housing_log);
//      detectorRegion->AddRootLogicalVolume(delta_housing_log);
//      detectorRegion->AddRootLogicalVolume(gamma_housing_log);

        //========================================================================

    G4double fast_offset = -0.5 * fast_sensorThickness;

        G4double fast_posLeadBack = fast_offset
                                                     + fast_leadThicknessBack / 2.;

    G4double fast_posCeramic = fast_offset
                                                    + fast_leadThicknessBack
                                                    + fast_ceramicThickness / 2.;

    G4double fast_posWax = fast_offset
                                                + fast_leadThicknessBack
                                                + fast_ceramicThickness
```

```
                                                + detectorThickness
                                                + fast_waxThickness / 2.;

    G4double fast_posLeadFront = fast_offset
                                                                + fast_leadThicknessBack
                                                                + fast_ceramicThickness
                                                                + detectorThickness
                                                                + fast_waxThickness
                                                                + fast_leadThicknessFront /
2.;

        //================================================================================

    fast_leadFront = new G4PVPlacement(0,

G4ThreeVector(0,0,fast_posLeadFront),

"fast_LeadFront",

fast_leadFront_log,
                                                                        fast_housing,
                                                                        false,
                                                                        0,
                                                                        1);

        //--------------------------------------------------------------------------------

    fast_wax = new G4PVPlacement(0,

G4ThreeVector(0,0,fast_posWax),
                                                                "fast_Wax",
                                                                fast_wax_log,
                                                                fast_housing,
                                                                false,
                                                                0,
                                                                1);

        //--------------------------------------------------------------------------------

    fast_ceramic = new G4PVPlacement(0,

G4ThreeVector(0,0,fast_posCeramic),
                                                                    "fast_Ceramic",
                                                                    fast_ceramic_log,
                                                                    fast_housing,
                                                                    false,
                                                                    0,
                                                                    1);

        //--------------------------------------------------------------------------------

    fast_leadBack = new G4PVPlacement(0,

G4ThreeVector(0,0,fast_posLeadBack),

"fast_LeadBack",

fast_leadBack_log,
                                                                        fast_housing,
                                                                        false,
                                                                        0,
                                                                        1);

        //--------------------------------------------------------------------------------
```

```
        G4double zPosFastSens = fast_sensorThickness/2. - fast_waxThickness
                                                - fast_leadThicknessFront
                                                                        -
detectorThickness/2.;

    physiFastSens = new G4PVPlacement(0,

G4ThreeVector(0,0,zPosFastSens),

"physiFastSens",
                                                                    logicFastSens,
                                                                    fast_housing,
                                                                    false,
                                                                    0,
                                                                    1);
//Albedo
        //============================================================================

        G4double albedo_hullThicknessFront      = 1.0*mm;
        G4double albedo_gapThickness            = 2.5*mm;
        G4double albedo_converterThickness      = 200.0*um;
        G4double albedo_ceramicThickness        = 0.5*mm;
        G4double albedo_hullThicknessBack       = 1.0*mm;
        G4double albedo_holeDiameter            = 5.0*mm;

        G4double albedo_sensorThickness = albedo_hullThicknessBack
                                                                        +
albedo_ceramicThickness
                                                                        +
detectorThickness
                                                                        +
albedo_converterThickness
                                                                        +
albedo_gapThickness
                                                                        +
albedo_hullThicknessFront;

        G4Box* albedo_housing_s = new G4Box("albedo_housing_s",

sensorWidthExt/2.,

sensorWidthExt/2.,

albedo_sensorThickness/2.);
        G4LogicalVolume*
        albedo_housing_log = new G4LogicalVolume(albedo_housing_s,

Cadmium,

"albedo_housing_log");
        detectorRegion->AddRootLogicalVolume(albedo_housing_log);
        //albedo_housing_log->SetVisAttributes(lead_vis);

        //----------------------------------------------------------------------------
        G4RotationMatrix* rotAlbedo = new G4RotationMatrix();
        rotAlbedo->rotateY(180.*deg);

    G4PVPlacement* albedo_housing =
                                                new G4PVPlacement(rotAlbedo,
#ifdef

G4ThreeVector(12.*cm,12.*cm,-15.*cm),
#else
```

```
                                                                                //
G4ThreeVector(0.,0.,PMMAPhantomSize.x()/2.

//                        +fast_sensorThickness/2.-5.*cm),

G4ThreeVector(0.,-10.*cm,0.),
#endif

"Albedo_Housing",

albedo_housing_log,

fPhysiPMMAPhantom,
                                                                                false,
                                                                                0);
//Albedo Sensor components
        //------------------------------------------------------------------------

    G4Box* albedo_hullFront_s = new G4Box("albedo_hullFront_s",

sensorWidth/2.,

sensorWidth/2.,

albedo_hullThicknessFront/2.);

        //------------------------------------------------------------------------

    G4Box* albedo_gap_s = new G4Box("albedo_gap_s",
                                                                        sensorWidth/2.,
                                                                        sensorWidth/2.,

albedo_gapThickness/2.);

        //------------------------------------------------------------------------

    G4Box* albedo_converter_s = new G4Box("albedo_converter_s",

sensorWidth/2.,

sensorWidth/2.,

albedo_converterThickness/2.);

        //------------------------------------------------------------------------

    G4Box* albedo_ceramic_s = new G4Box("albedo_ceramic_s",

sensorWidth/2.,

sensorWidth/2.,

albedo_ceramicThickness/2.);

        //------------------------------------------------------------------------

    G4Box* albedo_hullBack_s = new G4Box("albedo_hullBack_s",

sensorWidth/2.,

sensorWidth/2.,

albedo_hullThicknessBack/2.);
```

```
        //------------------------------------------------------------------------

    G4Tubs* albedo_hole_s = new G4Tubs("albedo_hole_s",
                                                                        0,
albedo_holeDiameter/2.,
albedo_hullThicknessBack/2.,
                                                                        0,2*pi);

        //------------------------------------------------------------------------

    G4LogicalVolume* albedo_hullFront_log =
                                                        new
G4LogicalVolume(albedo_hullFront_s,
Cadmium,
"albedo_hullFront_log");
        //albedo_hullFront_log->SetVisAttributes(lead_vis);


        //------------------------------------------------------------------------

        G4LogicalVolume* albedo_gap_log =
                                                        new G4LogicalVolume(albedo_gap_s,
Air,
"albedo_gap_log");
        //albedo_gap_log->SetVisAttributes(G4VisAttributes::Invisible);

        //------------------------------------------------------------------------

        G4LogicalVolume* albedo_converter_log =
                                                        new
G4LogicalVolume(albedo_converter_s,
LiF_CH2,
"albedo_converter_log");
        //albedo_converter_log->SetVisAttributes(converter_vis);

        //------------------------------------------------------------------------

        G4LogicalVolume* albedo_ceramic_log =
                                                        new
G4LogicalVolume(albedo_ceramic_s,
Ceramic,
"albedo_ceramic_log");
    //albedo_ceramic_log->SetVisAttributes(ceramic_vis);

        //------------------------------------------------------------------------

        G4LogicalVolume* albedo_hullBack_log =
                                                        new
G4LogicalVolume(albedo_hullBack_s,
Cadmium,
"albedo_hullBack_log");
    //albedo_hullBack_log->SetVisAttributes(lead_vis);
```

```
        //------------------------------------------------------------------------

    G4LogicalVolume* albedo_hole_log =
                                                          new
G4LogicalVolume(albedo_hole_s,

Air,

"albedo_hole_log");
    //albedo_hole_log->SetVisAttributes(hole_vis);

        //========================================================================


    G4double albedo_offset = -0.5 * albedo_sensorThickness;

        G4double albedo_posHullBack = albedo_offset
                                                                +
albedo_hullThicknessBack / 2.;

        G4double albedo_posCeramic  = albedo_offset
                                                                +
albedo_hullThicknessBack
                                                                + albedo_ceramicThickness
/ 2.;

    G4double albedo_posConverter = albedo_offset
                                                                 +
albedo_hullThicknessBack
                                                                 +
albedo_ceramicThickness
                                                                 + detectorThickness
                                                                 +
albedo_converterThickness / 2.;

    G4double albedo_posGap = albedo_offset
                                                + albedo_hullThicknessBack
                                                + albedo_ceramicThickness
                                                + detectorThickness
                                                + albedo_converterThickness
                                                + albedo_gapThickness / 2.;

    G4double albedo_posHullFront = albedo_offset
                                                                +
albedo_hullThicknessBack
                                                                +
albedo_ceramicThickness
                                                                + detectorThickness
                                                                +
albedo_converterThickness
                                                                + albedo_gapThickness
                                                                +
albedo_hullThicknessFront / 2.;

        //------------------------------------------------------------------------

    albedo_hullFront = new G4PVPlacement(0,

G4ThreeVector(0,

   0,

   albedo_posHullFront),
```

```
"albedo_HullFront",

albedo_hullFront_log,

albedo_housing,
                                                                    false,
                                                                    0);

    albedo_gap = new G4PVPlacement(0,

G4ThreeVector(0,0,albedo_posGap),
                                                          "albedo_Gap",
                                                          albedo_gap_log,
                                                          albedo_housing,
                                                          false,
                                                          0);

    albedo_converter = new G4PVPlacement(0,

G4ThreeVector(0,

   0,

   albedo_posConverter),

"albedo_Converter",

albedo_converter_log,

albedo_housing,
                                                                    false,
                                                                    0);

    albedo_ceramic = new G4PVPlacement(0,

G4ThreeVector(0,

 0,

 albedo_posCeramic),

"albedo_Ceramic",

albedo_ceramic_log,

albedo_housing,
                                                               false,
                                                               0);

    albedo_hullBack = new G4PVPlacement(0,

G4ThreeVector(0,

  0,

  albedo_posHullBack),

"albedo_HullBack",

albedo_hullBack_log,

albedo_housing,
                                                                  false,
```

```
                                                                        0);

    albedo_hole = new G4PVPlacement(0,
                                                                  G4ThreeVector(0,0,0),
                                                                      "albedo_Hole",
                                                                      albedo_hole_log,
                                                                      albedo_hullBack,
                                                                      false,
                                                                      0);


        //================================================================================

        G4String yDetRepName("DetRepY");
        G4VSolid* solYRepDet = new G4Box(yDetRepName,
                                                                  sensorWidth/2.,
                                                                  sensorWidth/2.,
                                                                  detectorThickness/2.);

        G4LogicalVolume* logYRepDet = new G4LogicalVolume(solYRepDet,

  fWater,

  yDetRepName);
        //G4PVReplica* yReplica =
        new G4PVReplica(yDetRepName,logYRepDet,logicFastSens,
                                                                       kYAxis,
1,sensorWidth);
        // X Slice
        G4String xDetRepName("DetRepX");
        G4VSolid* solXRepDet = new G4Box(xDetRepName,
                                                                  sensorWidth/2.,
                                                                  sensorWidth/2.,
                                                                  detectorThickness/2.);

        G4LogicalVolume* logXRepDet = new G4LogicalVolume(solXRepDet,

  fWater,

  xDetRepName);
        //G4PVReplica* xReplica =
        new G4PVReplica(xDetRepName,logXRepDet,logYRepDet,kXAxis,1,sensorWidth);

//---------------------------------------------------------------------------

        //.................................
        // Voxel solid and logical volumes
        //.................................
        // Z Slice
        G4String zDetVoxName("FastSens");
        G4VSolid* solDetVoxel = new G4Box(zDetVoxName,
                                                                  sensorWidth/2.,
                                                                  sensorWidth/2.,
                                                                  layerThickness/2.);

        fLVPhantomSens = new G4LogicalVolume(solDetVoxel,fWater,zDetVoxName);

        std::vector<G4Material*> fastSensMat(2,Silicon);
        fastSensMat[1]=Silicon;

        fNz=numberOfLayers;

        G4ThreeVector detSize(sensorWidth,sensorWidth,layerThickness);
        NestedPhantomParameterisation* paramFastSens
```

```
                    = new NestedPhantomParameterisation(detSize/2.,
                                                                    fNz,
fastSensMat);

        //G4VPhysicalVolume * physiPhantomSens =
    new G4PVParameterised("FastSens",     // their name
                          fLVPhantomSens,    // their logical volume
                          logXRepDet,           // Mother logical volume
                          kUndefined,        // Are placed along this axis
                          numberOfLayers,           // Number of cells
                          paramFastSens);     // Parameterisation.

    fUSDVolumes.push_back(fLVPhantomSens);
    fUSDNames.push_back(zDetVoxName);

        //=============================================================================

    G4Box* solidAlbedoSens = new G4Box("solidAlbedoSens",
                                                                    sensorWidth/
2.,
                                                                    sensorWidth/
2.,
detectorThickness/2.);

    G4LogicalVolume* logicAlbedoSens = new G4LogicalVolume(solidAlbedoSens,
                                                                    Silicon,
"logicAlbedoSens");

        G4double zPosAlbedoSens = albedo_sensorThickness/2.
                                                    - albedo_converterThickness
                                                    - albedo_gapThickness
                                  - albedo_hullThicknessFront
                                                    - detectorThickness/2.;

    physiAlbedoSens = new G4PVPlacement(0,
                                                G4ThreeVector(0,0,zPosAlbedoSens),
                                                "physiAlbedoSens",
                                                logicAlbedoSens,
                                                albedo_housing,
                                                false,
                                                0,
                                                1);

        G4String yADetRepName("ADetRepY");
        G4VSolid* solYRepADet = new G4Box(yADetRepName,
                                      sensorWidth/2.,
                                  sensorWidth/2.,
                                  detectorThickness/2.);

    G4LogicalVolume* logYRepADet = new G4LogicalVolume(solYRepADet,
                                                fWater,
                                                yDetRepName);

                              new G4PVReplica(yADetRepName,
                                                        logYRepADet,
                                                        logicAlbedoSens,
                                                        kYAxis,
                                                        1,
                                                        sensorWidth);

    G4String xADetRepName("ADetRepX");
```

```
    G4VSolid* solXRepADet = new G4Box(xADetRepName,
                                      sensorWidth/2.,
                                      sensorWidth/2.,
                                      detectorThickness/2.);

    G4LogicalVolume* logXRepADet = new G4LogicalVolume(solXRepADet,
                                                       fWater,
                                                       xDetRepName);
                      new G4PVReplica(xADetRepName,
                                                                  logXRepADet,
                                                                  logYRepADet,
                                                                  kXAxis,
                                                                  1,
                                                                  sensorWidth);


        G4String zADetVoxName("AlbedoSens");
    G4VSolid* solADetVoxel = new G4Box(zDetVoxName,
                                      sensorWidth/2.,
                                     sensorWidth/2.,
                                    layerThickness/2.);
        G4LogicalVolume*
    fAlbedoZSens = new G4LogicalVolume(solADetVoxel,fWater,zADetVoxName);

    std::vector<G4Material*> albedoSensMat(2,Silicon);
    albedoSensMat[1]=Silicon;

    fNz=numberOfLayers;

    G4ThreeVector adetSize(sensorWidth,sensorWidth,layerThickness);
    NestedPhantomParameterisation* paramAlbedoSens
                              = new NestedPhantomParameterisation(adetSize/2.,
                                                                  fNz,
                                                                  albedoSensMat);

        new G4PVParameterised("AlbedoSens",     // their name
                              fAlbedoZSens,
                              logXRepADet,
                                    kUndefined,
                                      numberOfLayers,
                                      paramAlbedoSens);

        fUSDVolumes.push_back(fAlbedoZSens);
        fUSDNames.push_back(zADetVoxName);

        return physiWorld;
}

void DetectorConstruction::ConstructSDandField() {

        //=================================================
        // Sensitive detectors : MultiFunctionalDetector
        //=================================================
        //
        //  Sensitive Detector Manager.
        G4SDManager* pSDman = G4SDManager::GetSDMpointer();

        std::vector<G4LogicalVolume*>::iterator LV_i = fUSDVolumes.begin();
        std::vector<G4LogicalVolume*>::iterator LV_n = fUSDVolumes.end();
        for(; LV_i != LV_n; ++LV_i)
        {
                FluenceEnergyDistributionSD* EDistri =
                      new FluenceEnergyDistributionSD((*LV_i)->GetName(),
                                                                    fNx,fNy,fNz,0);
```

```
                pSDman->AddNewDetector(EDistri);
                (*LV_i)->SetSensitiveDetector(EDistri);
        }
}
```