Turning GEANT Inside-Out:

A Python-GEANT4 Bridge which Provides Flexibility and Interactivity

Marcus Mendenhall
Vanderbilt University FEL Center

Overview

- GEANT4 provides an excellent low-level toolkit for fast computation of transport of moving particles
- Base language of GEANT4 is C++, which is needed for speed
- C++ is a relatively lousy language for data management
  - Much effort is needed to explicitly handle polymorphic data
  - Attention is needed for object creation, deletion, and ownership
  - All interaction done through limited Messenger classes
- Modern object-based scripting languages provide much better functionality for setup and management

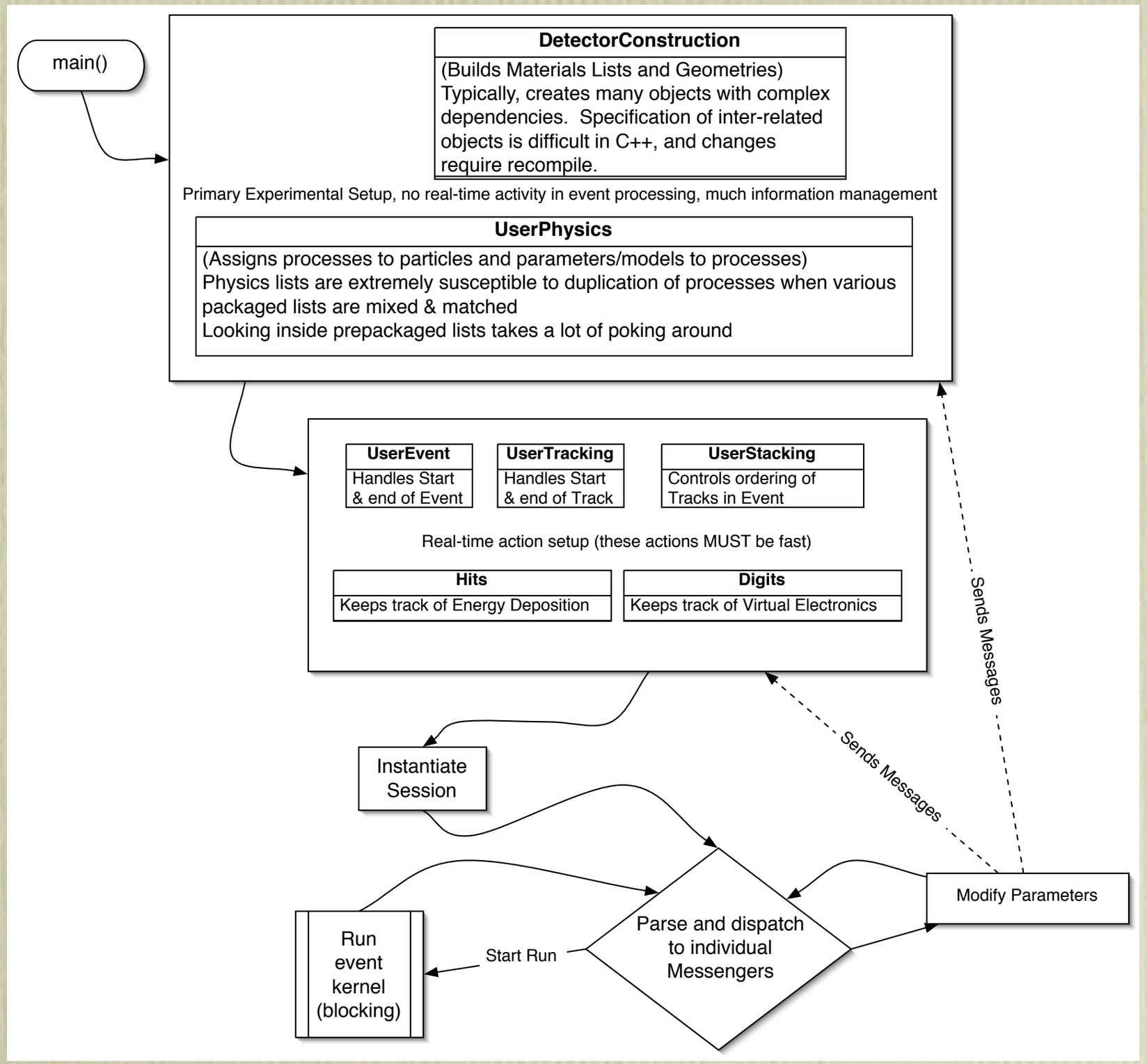
Architecture

- Instead of compiled main() doing setup, then creating a Session to handle user interaction, let the interactive interpreter run the show
- Use Python shell, with all its power, as the interpreter
- Use SWIG to wrap all needed GEANT4 and User classes, providing direct access to methods, instead of using hand-written Messenger classes to expose pieces
- Extend SWIG wrappers with specialized tools
- Use Python threads capabilities to allow keyboard access to multithreaded operations
- Use Python reference-counting object management to take care of all housekeeping

So What is SWIG?

- The Standard Wrapper Interface Generator
  - Automatically translates C and C++ header files into complex wrapper functions which allow various scripting languages to access compiled code.
  - Takes care of much data-type conversion, class management, etc.
  - Knows a lot about C++ class structure.
  - Needs only a very little bit of help typically to wrap most code.
  - Can be given more information to generate helper code to fill in gaps where the automatic wrappers are inconvenient.
- The long answer: see http://swig.sourceforge.net

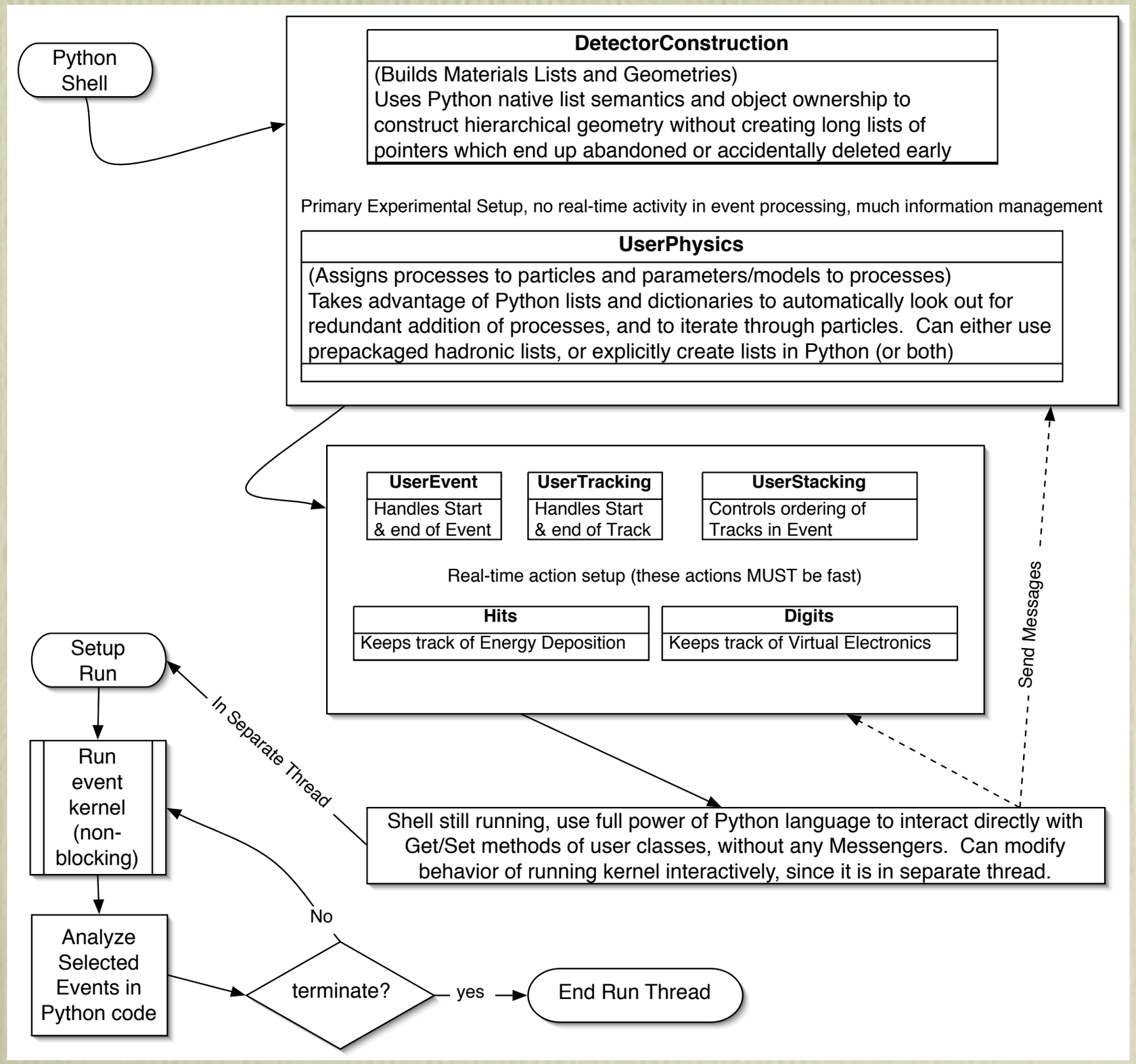
The Canonical GEANT World



Typical Old-Style Material Setup

```
//Air
G4Element* elN = new G4Element("Nitrogen", "N", z=7., a= 14.01*g/mole);
G4Element* elO = new G4Element("Oxygen"  , "O", z=8., a= 16.00*g/mole);

//G4Element* elGd = new G4Element("Gadolinium", "Gd", z=64.0, a=157.25*g/mole);
G4Element* elI = new G4Element("Iodine", "I", z=53.0, a=126.904*g/mole);
G4Element* elH = new G4Element("Hydrogen", "H", z=1., a= 1.0*g/mole);
G4Element* elC = new G4Element("Carbon"  , "C", z=6., a= 12.00*g/mole);
G4Element* elCa = new G4Element("Calcium"  , "Ca", z=20., a= 40.078*g/mole);
G4Element* elCl = new G4Element("Chlorine"  , "Cl", z=17., a= 35.4527*g/mole);

G4Material* Air = new G4Material("Air", density= 1.29*mg/cm3, nel=2,
                         kStateGas, 300.0*kelvin, 1.0*atmosphere);
Air->AddElement(elN, 70*perCent);
Air->AddElement(elO, 30*perCent);

// Tissue (Br12)
G4Material* br12 = new G4Material("BR12", density= 0.98*g/cm3,nComponents=6,
                            kStateSolid, 300.0*kelvin, 1.0*atmosphere);
br12->AddElement(elC, 0.7037);
br12->AddElement(elO, 0.1693);
br12->AddElement(elH, 0.0961);
br12->AddElement(elN, 0.0194);
br12->AddElement(elCa, 0.0086);
br12->AddElement(elCl, 0.0020);

// Iodinated Tissue (5% DNA Thymidine replacement = 1.3 mg/g, guess 20% for here)
G4Material* br12iod = new G4Material("iodinated BR12", density= 0.98*g/cm3,nComponents=2,
                           kStateSolid, 300.0*kelvin, 1.0*atmosphere);

br12iod->AddMaterial(br12, 0.995);
br12iod->AddElement(elI,0.005);
```

The Pythonized GEANT World



General Setup (was main())

```
G4Sys=G4System()
print "Geant4 Core Version: "+G4Core.GetVersionString()
ranecu=G4Core.RanecuEngine()
G4Core.HepRandom.setTheEngine(ranecu)

myDetector=G4Core.PythonDetectorConstruction()
myEvent=G4Core.ExN02EventAction()
myGenerator=G4Core.ExN02PrimaryGeneratorAction()
myRunAction=G4Core.SimpleRunAction()
myTrackingAction=G4Core.T01UserTrackingAction()
mySteppingAction=G4Core.ExN02SteppingAction()
visManager = G4Core.ExN02VisManager()
G4Sys.SetActions(detector=myDetector, event=myEvent, generator=myGenerator,
      run=myRunAction, tracking=myTrackingAction, stepping=mySteppingAction, vis=visManager)

baseSD=G4Core.ExN02TrackerSD(G4String("BaselineSD"))
G4Core.AddNewSensitiveDetector(baseSD)
tumorSD=G4Core.ExN02TrackerSD(G4String("TumorSD"))
G4Core.AddNewSensitiveDetector(tumorSD)

targetParams=dict(
   tumorFullThickness=20.*millimeter,
   worldFullHeight=100.0*millimeter,
   skinFullThickness=150*millimeter,
   tumorCenterDepth=20.0*millimeter,
   boneCenterDepth=75.0*millimeter,
   boneFullThickness=30.0*millimeter,
   iodineFrac=0.5*perCent,
   baseSD=baseSD, tumorSD=tumorSD)
worldFullLength, actualPositionTumor =CreateGeometry(**targetParams)

import RadiationPhysics
EM=RadiationPhysics.PenelopePhysics(secondaryCut=2.0*keV, useAuger=0, useFluorescence=1).Construct()
had=RadiationPhysics.DumbHadronPhysics().Construct()
G4Core.G4ProductionCutsTable.GetProductionCutsTable().SetEnergyRange(0.1*keV, 1.0*GeV)
G4Sys.SetActions(physics=EM) #just a formality
G4Sys.Initialize()
```

Typical Python Materials Setup

Arbitrary Length Lists

String formatting

Real keyword arguments!

```
BR12= MaterialWeight("BR12", density= 0.98*gram/cm3,
   elements_list=(
      ("C", 0.7037), ("O", 0.1693), ("H", 0.0961), ("N", 0.0194), ("Ca", 0.0086), ("Cl", 0.0020)
   )
)

BR12iod = MaterialMixture("BR12+%.1f%% Iodine"%(iodineFrac/perCent),
   density=0.98*gram/cm3,
   materials_list=(
      (BR12, 1.0-iodineFrac),
      ("I", iodineFrac)
   )
)

Bone=MaterialWeight("icru-44 bone", density= 1.92*gram/cm3,
   elements_list=(
      ("H", 0.034), ("C", 0.155), ("N", 0.042), ("O", 0.435),
      ("Na", 0.001), ("Mg", 0.002), ("P", 0.103), ("S", 0.003),
      ("Ca", 0.225)
   )
)
```

Geometry Setup in Python

Helper class to manage ownership

```
solidTumor=UnionSolid("Bumpy Tumor",
   G4Tubs(G4String("Dosed tumor"),
            0., tumorFullThickness/2.0, worldFullHeight,
            0.0*degree, 360.0*degree
         ),
   G4Tubs(G4String("Dosed tumor bump"),
            0., tumorFullThickness/4.0, worldFullHeight,
            0.0*degree, 360.0*degree
         ),
   pos=(0., tumorFullThickness/2.0, 0.)
)

logicTumor=LogicalVolume(solidTumor,
   material=BR12iod, name="tumor", sensitive=tumorSD, color=(0,1,1)
)

logicTumor.SetForceSolid(1)

relativePositionTumor = Numeric.array((-skinFullThickness/2.0+tumorCenterDepth, 0, 0))
actualPositionTumor=Numeric.dot(RotationY(-90.0*degree), relativePositionTumor)

Placement(logicTumor, "tumor", logicSkin, pos=relativePositionTumor)
```

Easy attachment of SensitiveDetectors

Inline VisAttributes

G4RunManager, Exposed!

- Expose a little more of the RunManager so that more of the run setup and termination can be executed in Python
  - This allows one to run some events, move the ParticleGun, and then run more, for example, to create diffuse (or other) sources
  - Also, can use UserStacking or UserEvent actions to terminate the run reversibly when an event arises which is 'interesting'. This permits fast but simple pre-analysis of events in compiled code, followed by slower, flexible analysis in Python code. Run can be continued after 'interesting' event is processed.

Run with AIDA & OpenDX

```
def run_series(thetalist=(-20,0,20), events=100000, sdBase=None, sdTumor=None):
    global _running
    _running=1
    try:
        if sdBase and sdTumor:
            runcode=time.strftime("%Y%m%d.%H%M%S")
            import AIDASupport #only import AIDA if we really need it!
            import AIDA
            tree=AIDASupport.AIDATree("", createNew=1, options="compress=yes")
            dx=worldFullLength*0.5
            chans=300
            histfact=tree.GetHistogramFactory()
            dc1Hits = histfact.createHistogram3D("Ionization_Map_1",1,
                -dx,dx,chans,-dx,dx,chans,-dx,dx)
            sdBase.SetXYZHistogram(dc1Hits)
            dc2Hits = histfact.createHistogram3D("Ionization_Map_2",1,
                -dx,dx,chans,-dx,dx,chans,-dx,dx)
            sdTumor.SetXYZHistogram(dc2Hits)
        else:
            tree=None

        G4Sys.SetupRun()

        for theta in thetalist:
            if not _running: break #someone else is sending us a message
            gunPos=Numeric.array((0,0,-1.1*targetParams['skinFullThickness']/2.0))
            gunPos=Numeric.dot(RotationX(theta*degree), gunPos)
            myGenerator.SetGunPosition(Vector(gunPos))
            myGenerator.SetGunAxis(Vector(actualPositionTumor-gunPos))
            G4Sys.runMgr.DoEventLoopThreaded(events)

        G4Sys.FinishRun()

        if tree:
            sdBase.UnsetHistogram()
            sdTumor.UnsetHistogram()
            hist2d1=histfact.projectionYZ("hist2d1", dc1Hits) #project down
            hist2d2=histfact.projectionYZ("hist2d2", dc2Hits) #project down
            final_plain=histfact.add("unenhanced", hist2d1, hist2d2)
            scale=2.0
            hist2d2.scale(scale) #adjust for DER
            final_der=histfact.add("DER = %.1f enhanced"%scale, hist2d1, hist2d2)

            for hist, basename in ( (final_der, "DER_%.1f"%scale),
                    (final_plain, "unenhanced") ):
                vf=AIDA.vectorDouble()
                AIDA.Convert2DHistToVector(hist, vf)
                histbins=Numeric.array(vf, Numeric.Float32)

                datafilename=basename+"."+runcode+"."
                savehistfile=file(datafilename+"binary","w")
                savehistfile.write(histbins.tostring())
                savehistfile.close()
                xax=hist.xAxis()
                yax=hist.yAxis()

                dxheader=create_dx_header_2d(datafilename+"binary",
                        xax.bins(), yax.bins(),
                        xax.binLowerEdge(0), xax.binWidth(0),
                        yax.binLowerEdge(0), yax.binWidth(0))
                gf=file(datafilename+"general","w")
                gf.write(dxheader)
                gf.close()

    finally:
        _running=0
```

How Hard is This?

- Simplest Case
  - Rewrite main() as SetupGeant() so it ends where the session starts
  - Run Gnumake to build libfoo.a where foo is the project name
  - Generate a typically 20-line SWIG declaration to expose interesting parts of user classes and run SWIG
  - Run python setup.py to let distutils build the modules
  - Enjoy!
- More real Pythonization
  - Move Materials and Detector construction into Python
  - Move Physics Lists into Python
  - Eliminate Messenger classes, replacing with native Python access
  - Move bits of RunManager loop into Python for event analysis

Why?

- Code Maintainability
  - Way too much effort required to do complex hierarchical object management in C++. Almost every change results in memory leaks.
  - Much less attention required to declaring types everywhere.
  - Resulting code is much shorter, since Python does much of the work.
- Code Flexibility
  - Only the gory low-level details of event handling, *etc.* live in compiled code. Everything else can be changed in the interpreted layer.
  - The SWIG-wrapped classes automatically expose all public class methods, instead of only a few which Messengers usually expose.
  - File I/O and management is very easy to change as needed.
- Speed of Development is MUCH better.

Why? again: The Python Library

- The Python library provides extensive XML file parsing capabilities, which will ultimately make interchange with CAD easier.
- The library provides extensive web-interaction tools, potentially making it easy to provide access to your simulation via web-based tools, so remote users can interact with your local experiment simulator to test out concepts

Conclusion (?)

- The conclusion is that this is not at all concluded!
- The presentation is really a demonstration of a possible future framework for much GEANT work.
- It is easily expandable to use new C++ classes, since SWIG automatically does the work needed.
- It is easily expandable on the Python end to provide extensive new management and analysis capabilities, since Python is both a very powerful scripting language and has an excellent library of tools bundled with it.
- We would like to work closely with the core development team to see how to integrate this into a (not-too-far) future GEANT toolkit.