Installation Instructions

Material Properties

Before you can use the G4S1Light and G4S2Light physics processes, you need to add new material properties to your noble elements. Since everyone's Geant4 simulation is set up slightly differently, with different file names, directory structure, etc. we could not create a one-size-fits-all automatic installer. Instead, here in this readme we offer you blocks of code which you must copy and paste into the appropriate places within your own simulation's code, changing variable names where/if necessary.

First, open the source code for your materials list. At some point you must already have something (obviously, your variable names may differ from these example lines of code) along the lines of:

```
    inline G4MaterialPropertiesTable *LXeTable() { return liquidXeMat; };      //may be in your header
….
    G4MaterialPropertiesTable *liquidXeMat;    //may be in your header file
….
….
    delete liquidXeMat;   //must be in your destructor method!
….
    // you almost certainly already have the following, which is just a definition for liquid xenon
    G4Element *natXe = new G4Element( "Natural Xe", "natXe", 9 );
    natXe->AddIsotope( Xe124, 0.09*perCent );
    natXe->AddIsotope( Xe126, 0.09*perCent );
    natXe->AddIsotope( Xe128, 1.92*perCent );
    natXe->AddIsotope( Xe129, 26.44*perCent );
    natXe->AddIsotope( Xe130, 4.08*perCent );
    natXe->AddIsotope( Xe131, 21.18*perCent );
    natXe->AddIsotope( Xe132, 26.89*perCent );
    natXe->AddIsotope( Xe134, 10.44*perCent );
    natXe->AddIsotope( Xe136, 8.87*perCent );

    liquidXe = new G4Material( "liquidXe", 2.953*g/cm3, 1 ); //you already have this in your LXe sim
    liquidXe->AddElement( natXe, 1 );
….
    liquidXeMat = new G4MaterialPropertiesTable(); //the actual initialization
```

The last lines above are where you first define liquid xenon as a material, complete with a proper Geant4 material properties table. Next, if you don't already have it, you need to define an array of photon energies, which you probably already have, for the purpose of defining an index of refraction as a function of photon wavelength, but, just in case, here is an example:

```
    // this first block (or something a lot like it) may already lie in a header file
    G4double *photonWavelengths;
    G4double *photonEnergies;

    // may be in your header file, or split between code and header so look for it first
    G4int num_pp = 22; //number of photon energies/wavelengths we will define in our example below
```

```
const G4int NUM_PP = num_pp;
photonWavelengths = new G4double[NUM_PP]; photonEnergies = new G4double[NUM_PP];
photonWavelengths[0] = 144.5;
photonWavelengths[1] = 175.8641;
photonWavelengths[2] = 177.6278;
photonWavelengths[3] = 179.4272;
photonWavelengths[4] = 193.6;
photonWavelengths[5] = 250.3;
photonWavelengths[6] = 303.4;
photonWavelengths[7] = 340.4;
photonWavelengths[8] = 410.2;
photonWavelengths[9] = 467.8;
photonWavelengths[10] = 508.6;
photonWavelengths[11] = 546.1;
photonWavelengths[12] = 627.8;
photonWavelengths[13] = 706.5;
photonWavelengths[14] = 766.5;
photonWavelengths[15] = 844.7;
photonWavelengths[16] = 1000.0;
photonWavelengths[17] = 1300.0;
photonWavelengths[18] = 1529.6;
photonWavelengths[19] = 1600.0;
photonWavelengths[20] = 1800.0;
photonWavelengths[21] = 2058.2;
for( G4int i=0; i<NUM_PP; i++ )
    photonEnergies[i] =
            (4.13566743E-15*299792458/(photonWavelengths[i]*1.E-9))*eV;
```

Now, we’ve established two important things, NUM_PP and the photonEnergies array. **Don’t forget to define indices of refraction for these wavelengths**. Here we supply an index of refraction definition you can copy over instead of being forced to tediously generate your own (provided by Scott Stephenson).

```
//refractive index values taken from: 'New approach to the calculation
//of the refractive index of liquid and solid xenon' (some values merely estimated)
//https://ir.kochi-u.ac.jp/dspace/bitstream/10126/4658/1/Hitachi2005JCP234508.pdf***
LXeRefractiveIndex[0] = 1.82;
LXeRefractiveIndex[1] = 1.663;// originally ~1.63 <-> possible shift needed based on temperature?
LXeRefractiveIndex[2] = 1.648;// originally ~1.61
LXeRefractiveIndex[3] = 1.637;// originally ~1.58
LXeRefractiveIndex[4] = 1.576;
LXeRefractiveIndex[5] = 1.468;
LXeRefractiveIndex[6] = 1.429;
LXeRefractiveIndex[7] = 1.415;
LXeRefractiveIndex[8] = 1.404;
LXeRefractiveIndex[9] = 1.3982;
LXeRefractiveIndex[10] = 1.3951;
LXeRefractiveIndex[11] = 1.392;
LXeRefractiveIndex[12] = 1.3879;
```

***Other possible sources are A. C. Sinnock and B. L. Smith, "Refractive Indices of the Condensed Inert Gasses", Physical Review 181(3) (1969) p.1297 and L.M. Barkov et al., NIM A379 (1996), 482. You have to avoid the G4Exception where wavelength goes out of range, by defining index.

```
LXeRefractiveIndex[13] = 1.3865;
LXeRefractiveIndex[14] = 1.3861;
LXeRefractiveIndex[15] = 1.386;
LXeRefractiveIndex[16] = 1.386;
LXeRefractiveIndex[17] = 1.385;
LXeRefractiveIndex[18] = 1.385;
LXeRefractiveIndex[19] = 1.384;
LXeRefractiveIndex[20] = 1.384;
LXeRefractiveIndex[21] = 1.383;
```

Note that if you have any reflective surfaces in your simulation that you will have to define a reflection coefficient for each wavelength, or at least for the minimum and maximum wavelengths (making it implicitly constant or linear). Otherwise, photons will just stop propagation once they reach the surface if nothing is defined. After you've got all your optical properties settled, set your electric fields next:

```
    liquidXeMat->AddConstProperty( "ELECTRICFIELD", 0*volt/cm ); //for missed nooks and crannies

    liquidXeMat->AddConstProperty( "ELECTRICFIELDSURFACE", 0*volt/cm );
    liquidXeMat->AddConstProperty( "ELECTRICFIELDGATE", 0*volt/cm );
    liquidXeMat->AddConstProperty( "ELECTRICFIELDCATHODE", 0*volt/cm );
    liquidXeMat->AddConstProperty( "ELECTRICFIELDBOTTOM", 0*volt/cm );
```

You can enter in any units you like that are accepted by Geant4 (V/cm, kV/cm, etc., with proper G4 names). Zero is not the only valid field, as it is only an example above. Put in any value you like for your one or two-phase detector. You will be forced to re-compile your code whenever changing the number, but we figure this is not a major inconvenience if your detector mainly sits at one field. You, the end-user, should be able to make this a run-time command on your own. This is not a true, Geant4 style electric field with accompanying physics, as this proved unnecessarily complicated for NEST, and you don't need one to get a good functioning drift occurring for your ionization electrons. The reason is NEST simply generates all ionization electrons with perfectly opposite-of-field-pointing momenta right off the bat, for simplicity, if a non-zero electric field is detected by the G4S1Light process (line 864).

There is now a new option to set four different liquid electric fields and three different gas fields for two-phase detectors, for full Monte Carlo gamma-X investigations. If you want you can set the coordinates at the top of G4S1Light.hh all to zero, and NEST defaults to "ELECTRICFIELD, " which is also the field for any unknown point that got missed in the field definitions. For gas, the names are:

```
    gasXeMat->AddConstProperty( "ELECTRICFIELD", 0*volt/cm ); //for missed nooks and crannies

    gasXeMat->AddConstProperty( "ELECTRICFIELDWINDOW", 0*volt/cm );
    gasXeMat->AddConstProperty( "ELECTRICFIELDTOP", 0*volt/cm );
    gasXeMat->AddConstProperty( "ELECTRICFIELDANODE", 0*volt/cm );
```

The setting of the electric field determines the amount of the quenching of the number of S1 scintillation photons with increasing electric field magnitude as the recombination probability decreases, with the amount of ionization electrons generated using NEST correspondingly increasing. NEST takes care of all of that for you, so you can dial in almost any field your heart desires (see Known Bugs section of the NEST Companion for exceptions), just like you can send any particles in that you want.

Last, but not least, we initialize the number of interaction sites, and then save everything with a ‘Set’:

```
    liquidXeMat->AddConstProperty( "TOTALNUM_INT_SITES", -1 );
    liquidXe->SetMaterialPropertiesTable( liquidXeMat ); //note liquidXe not liquidXeMat at start of line
```

For a two-phase detector, repeat the process above for gas. Also, xenon is only an example element.

Physics Lists

You need to add the 6 files G4S1Light.cc, G4S1Light.hh, G4S2Light.cc, G4S2Light.hh, G4ThermalElectron.cc, and G4ThermalElectron.hh to the source and header sub-directories, as appropriate, of the physicslist directory of your Geant4 simulation. Make sure your makefile is set up in such a way that it looks for and finds all of the *.cc/*.hh files on its own, for compilation, during the making of the executable of your simulation. But you still need to change some function calls and header includes. Most likely, you will have a header file for your optical physics list. Look for all instances of G4Scintillation and globally find and replace that term with G4S1Light.

Usually, you would need to make 3 replacements (your variable name for G4S1Light* is free to differ):

```
    #include "G4Scintillation.hh"
....
    G4Scintillation *theScintProcess;
....
    G4Scintillation *GetScintillation() { return theScintProcess; };
```

become

```
    #include "G4S1Light.hh"
....
    G4S1Light *theScintProcess;
....
    G4S1Light *GetScintillation() { return theScintProcess; };
```

That was for the header. Now, the source code for your optical physics needs changing:

```
    theScintProcess = new G4Scintillation();
....
    if( theScintProcess->IsApplicable(*particle) ) {
        pManager->AddProcess(theScintProcess);
        pManager->SetProcessOrderingToLast(theScintProcess,idxAtRest);
        pManager->SetProcessOrderingToLast(theScintProcess,idxPostStep);
    }
```

becomes

```
    theScintProcess = new G4S1Light();
....
    if( theScintProcess->IsApplicable(*particle) )
        pManager->AddProcess(theScintProcess,ordDefault,ordInActive,ordDefault);
```

The NEST scintillation process is applied to every energy deposition which occurs in your noble, but looks out for zero-energy deposition steps. The above sets up G4S1Light as a RestDiscrete physics process using the so-called "magic numbers" of Geant4 (ord...) This is also a good place to add the line

```
    theScintProcess->SetScintillationYieldFactor(1.);
```

Make this say zero instead if you want to turn NEST off without painstakingly uninstalling it. Use a G4 messenger class to have a run-time flag for making this 0 or 1. The reason this is not a Boolean variable nor even an int is because it is used inside of G4S1Light to set a YieldFactor less than 1 (but greater than 0 of course) for nuclear recoil. (In other words, it is used for the Lindhard factor L.)

You can set whether secondaries from NEST's G4S1Light are tracked first, or, all of the other particles flying around already in your simulation are tracked first, with

```
    theScintProcess->SetTrackSecondariesFirst(true);
```

OR

```
    theScintProcess->SetTrackSecondariesFirst(false);
```

However, since NEST is designed to make scintillation only when all tracks from one run are complete, this setting should have no effect, and altering it is not recommended (or, leave it false).

Repeat all of the above for G4S2Light now if you want S2 light. Don't forget to have a different variable name other than theScintProcess if that's what you used for S1. (I use theLuminProcess for S2.) Also, for S2, you need to add #include G4ThermalElectron.hh

To maximize the accuracy of your simulation, it is recommended that you also open up your main physics list source code and change the cut-off value for distances to be

```
    shortCutValue = 1*nm;
.....
    SetCutsShort();
```

where in the function SetCutsShort you set the same short cut for all particles. Or, something else ridiculously, infinitesimally small. Don't worry about slowing down your simulation and bogging down your computer, as Geant4 does not have infinite accuracy, so you just bottom out. You can actually get away with a higher value, but above O(1um), extreme inaccuracy will begin to creep into your simulation, where, for example, only discrete-energy electron recoils can occur given a certain parent gamma. To be safe, keep it low unless it really slows things down.

If you really want to get the most out of NEST at lower energies, then you should also activate Auger electrons. This document cannot possibly tell you every single way to do this, given all the different versions of Geant out there, and different possible low-energy electromagnetic physics lists, but if you are utilizing the G4EmLivermore physics list for your sim, and are running Geant4.9.4 through patch 4, then you can go to the same place, line 195 of G4EmLivermorePhysics.cc, and add the line

```
    theLivermorePhotoElectricModel->ActivateAuger( true );   //(you shouldn't need any new variables)
```

You can find G4EmLivermorePhysics.cc in the source/physics_lists/builders/src/ directory of your Geant4 install. However, you do not need to re-compile Geant to make the change stick. Instead, just copy G4EmLivermorePhysics.cc into the source directory of your physics list and change that instead. Not doing this means the ~35 keV dip in LXe γ light yield caused by a K-edge won't be as deep as it should be (read the first NEST JINST paper). Read the Geant4 forums for how to do this for your physics list and Geant4 version if different, but in general look for G4EmLivermorePhysics::ConstructProcess() (or, the constructor processor of whatever physics list you are using if it isn't Livermore's) and look inside the if-statement which begins with if (particleName == "gamma").

We do not recommended any electromagnetic physics list over others in terms of accuracy at this time, but if you wish to remain in sync the most with the NEST development team at the present time and receive help with your questions in the most timely fashion, then you should use Livermore, but nevertheless please feel free to explore the different physics lists and tell us how your results differ. We have spot-checked others and have found <10% difference in yields, except for Penelope.

After you've made all of the necessary alterations to your physics and material lists, you are ready to compile and link your Geant4 simulation with NEST installed! Run macros as you did before, with the general particle source (gps), for example, but now your results will be very physical, complete with electric field quenching of S1, S2 production, and realistic photon and electron yields for nuclear recoil events, correctly reduced. By virtue of being rest/discrete processes instead of just discrete as they had been before, G4S1Light and G4S2Light are now properly logged if you turn on tracking verbosity. You *will* be able to find them as physics processes listed in text output.

*****VERY IMPORTANT**: Do NOT just blindly use this readme and copy and paste code without careful thought. If you've already constructed a detector geometry for a noble element, a lot of code which I tell you to add already exists in your simulation almost for sure, though perhaps not in the same files, not in the same order, and under different names. Additionally, if you've implemented G4Scintillation or your own home-grown version of it, then you must carefully remove all references to it, and replace them with the correct new ones. This includes material properties like the scintillation yield, which, if you're not careful, you may fail to override with the NEST code, and you may end up with code which "makes," but yields incorrect results: your work function, singlet/triplet times, etc. can conflict with ours.

This may occur in a subtle fashion, with production of plots that look OK. That is why you should use the table and plots located at http://nest.physics.ucdavis.edu/site/?q=benchmarks to validate your install.

An example few lines of a Geant4 macro utilizing the gps command follow:

```
/gps/ion 54 131
/gps/energy 10 keV
/gps/position 0 0 0 mm
/gps/ang/type iso
```

(Make /gps/position valid for your detector.) Possibilities are endless: create NR, shoot x-rays, gamma rays, neutrons, electrons, alphas, protons, light and heavy ions of your choice, excited nuclei, etc. Do any monoenergetic scenario you want, use an energy histogram feature, or do radioactive sources. Do angles and positions like normal, and dial in electric field strengths in your material properties table.

For advanced configuration of all the bells and whistles of NEST, see the last page of WhatNew.pdf.