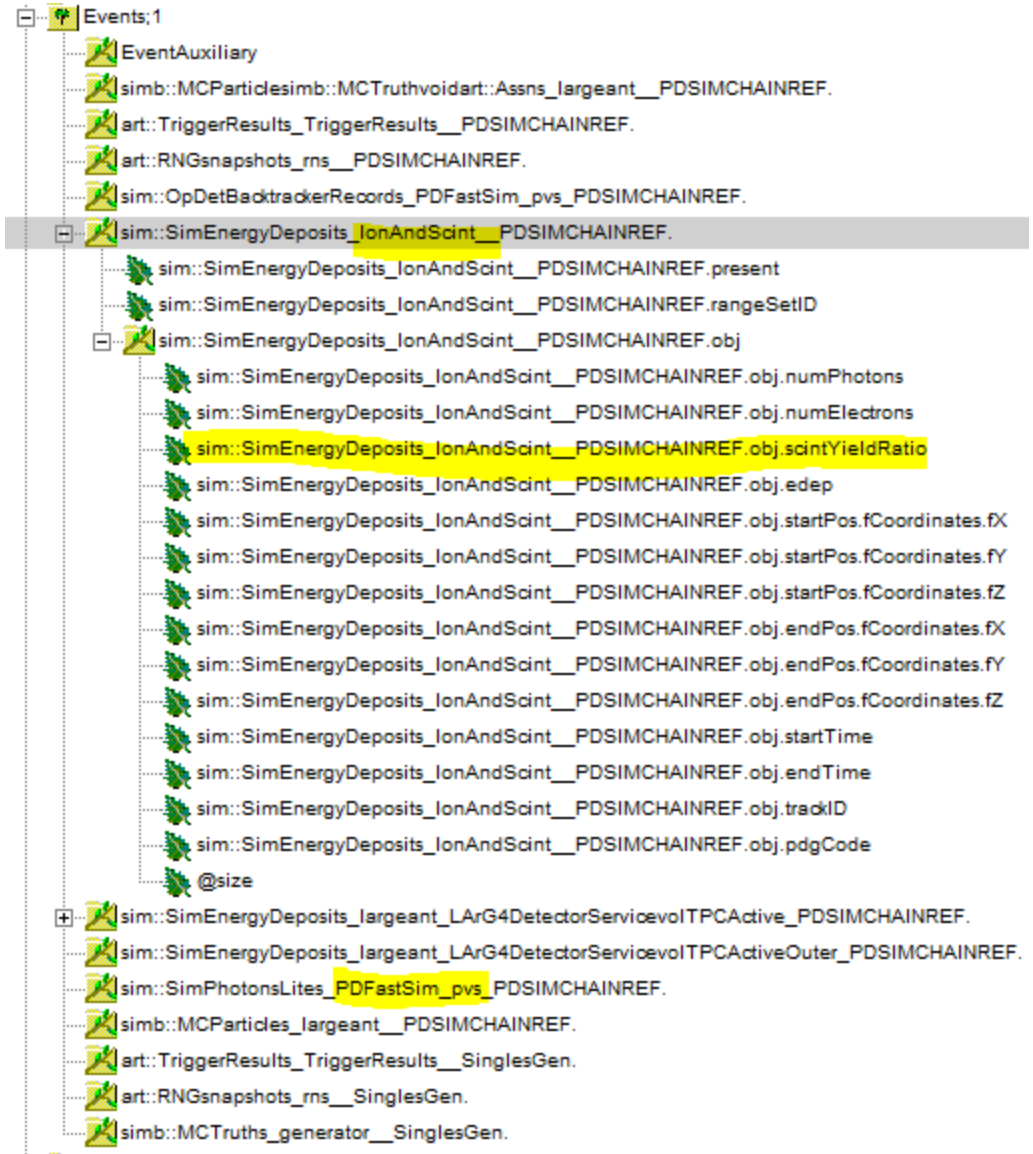


```
SimEnergyDeposit(int np = 0,
        int ne = 0,
        double sy = 0,
        double e = 0.,
        geo::Point_t start = {0.,0.,0.},
        geo::Point_t end = {0.,0.,0.},
        double t0 = 0.,
        double t1 = 0.,
        int id = 0,
        int pdg = 0)
  : numPhotons(np)
  , numElectrons(ne)
  , scintYieldRatio(sy)
  , edep(e)
  , startPos(start)
  , endPos(end)
  , startTime(t0)
  , endTime(t1)
  , trackID(id)
  , pdgCode(pdg)
{
} « end SimEnergyDeposit »
```

```
// Note that even if we store a value as float, we return
// it as double so the user doesn't have to think about
// precision issues.

int NumPhotons() const { return numPhotons; }
int NumFPhotons() const { return round(numPhotons * scintYieldRatio); }
int NumSPhotons() const { return round(numPhotons * (1.0 - scintYieldRatio)); }
int NumElectrons() const { return numElectrons; }
double ScintYieldRatio() const { return scintYieldRatio;}
double Energy() const { return edep; }
geo::Point_t Start() const { return { startPos.X(), startPos.Y(), startPos.Z() }; }
geo::Point_t End() const { return { endPos.X(), endPos.Y(), endPos.Z() }; }
```

```
IonAndScint:
{
    module_type:            "IonAndScint"
    SimulationLabel:        "largeant:LArG4DetectorServicevolTPCActive"
//    ISCalcAlg:              "NEST"
    ISCalcAlg:              "Separate"
}

PDFastSim:
{
   module_type:             "PDFastSimPVS"
//   module_type:             "PDFastSimPAR"

   SimulationLabel:         "IonAndScint"
   DoSlowComponent:         true
   ScintTimeTool:           @local::ScintTimeLAr
}
```

```
BEGIN_PROLOG

ScintTimeLAr:
{
    tool_type: ScintTimeLAr
    LogLevel:          1
    FastRisingTime:    0.0
    FastDecayTime:     6.0
    SlowRisingTime:    0.0
    SlowDecayTime:     1600.0
}

ScintTimeXeDopedLAr:
{
    tool_type: ScintTimeXeDopedLAr
    LogLevel:   1
}


END_PROLOG
```

art class tools

```
namespace phot
{
    class ScintTime
    {
    public:
        ScintTime();
        virtual void GenScintTime(bool is_fast, CLHEP::HepRandomEngine& engine) = 0;
        double GetScintTime() const    {return timing;}

    protected:
        double timing;
    };
}
```

```
namespace phot
{
    class ScintTimeLAr : public ScintTime
    {
    public:
        explicit ScintTimeLAr(fhicl::ParameterSet const& pset);
        void GenScintTime(bool is_fast, CLHEP::HepRandomEngine& engine);

    private:
        int          LogLevel;

        // parameters for the shape of argon scinitllation light time distribution
        double         SRTime;                       // PureLAr: rising time of slow LAr scinitllation;
        double         SDTime;                       // PureLAr: decay time of slow LAr scintillation;
        double         FRTime;                       // PureLAr: rising time of fast LAr scinitllation;
        double         FDTime;                       // PureLAr: decay time of fast LAr scintillation;

        // general functions
        double single_exp(double t, double tau2);
        double bi_exp(double t, double tau1, double tau2);
    };
}
```