

The BackTracker service must be, as of the Ides of March 2023, instantiated with a const\_cast:

```
    cheat::BackTrackerCore const* const_bt = gar::providerFrom<cheat::BackTracker>();
    cheat::BackTrackerCore*          bt = const_cast<cheat::BackTrackerCore*>(const_bt);
```

It also must be instantiated in the analyze method of the *art* plugin derived from EDAnalyzer, rather than the beginJob method.

The base class, BackTrackerCore, has a set of stdlib structures that are filled upon art's sPreProcessEvent. Then an example use is:

```
    auto RecoClusterHandle = e.getHandle< std::vector<rec::Cluster> >(fClusterLabel);
    if (!RecoClusterHandle) {
        throw cet::exception("anatest") << " No rec::Cluster branch."
        << " Line " << __LINE__ << " in file " << __FILE__ << std::endl;
    }
    for ( rec::Cluster cluster : *RecoClusterHandle ) {
        std::vector<std::pair<simb::MCParticle*,float>> whatMatches;
        whatMatches = bt->ClusterToMCParticles(&cluster);
        std::cout << "\nCluster No. " << cluster.getIDNumber() << " is made of MCParticles: " << std::endl;
        for (auto itr = whatMatches.begin(); itr!=whatMatches.end(); ++itr) {
            std::cout << "G4 track number " << itr->first->TrackID() << " \thas PDG code " <<
            itr->first->PdgCode() << " \tand its mother is G4 track " << itr->first->Mother()
            << " \tand energy fraction " << 100*(itr->second) << "%\n";
        }
    }
```

*The following should always work* -----

```
explicit BackTrackerCore(fhicl::ParameterSet const& pset);
~BackTrackerCore();
```

But of course. Default destructor.

```
void AdoptEveIdCalculator(simb::EveIdCalculator* ec);
```

One of the functions of the ParticleList class is to find Eve, the mother of all mother particles. The algorithm for determining Eve is declared with this method; the only algorithm that exists now is the nutools EmEveIdCalculator which moves up the tree until it finds a particle not produced by one of the GEANT processes {conv, LowEnConversion, Pair, compt, Compt, Brem, phot, Photo, Ion, Annihil}. Other functions of ParticleList are less useful for us.

*The following should work as long as there are MCParticle and MCTruth data products in the event, with an art::Assns between them* -----

```
simb::ParticleList* GetParticleList() const;
```

The backtracker uses the nutools ParticleList class (recently, in nug4) as an augmentation of std::map<int, simbMCParticle\*>, where the int is the GEANT trackID. This method returns a pointer to the instance of the ParticleList class which is in BackTrackerCore and is filled by the RebuildNoSC method.

```
simb::MCParticle* TrackIDToParticle(int const& id) const;
```

From a GEANT TrackID to an MCParticle. The reverse process is just MCParticle::TrackID(). Negative GEANT TrackIDs are believed to be EM shower particles. Will return nullptr on error condition.

```
simb::MCParticle* FindMother(simb::MCParticle* const p) const;
```

From an MCParticle to its mother particle. The MCParticle class has a Mother() method, but it returns a track ID not an MCParticle.

```
simb::MCParticle* FindEve(simb::MCParticle* const p) const;
```

From an MCParticle to its Eve MCParticle, using the adopted EveIdCalculator. Will return nullptr on error condition in TrackIDToParticle.

```
simb::MCParticle* FindTPCEve(simb::MCParticle* const p);
```

From an MCParticle to the progenitor MCParticle which exited the TPC or is a primary. Uses the GeometryCore::PointInMPP to find if one end of the particle is in the TPC and the other is not. Will return nullptr on error condition in TrackIDToParticle.

```
simb::MCParticle* FindTPCEve(int const trackID);
```

Signature using trackID mostly for call in CellIDToCalIDEs but may as well make it public. This is the method that prevents BackTrackerCore from being const. If DontChaseNeutrons in BackTracker.fcl is true, which is the default, a shower outside the TPC from a neutron will be traced back to that neutron even if it did not originate in the gas. The flag DontChaseNeutrons in BackTracker.fcl with default value true, will cause FoundTPCEve to stop chasing shower MCParticles up the tree if a neutron is found, regardless of the neutron's location in the detector.

```
simb::MCParticle* FindeCALEve(simb::MCParticle* const p);
simb::MCParticle* FindeCALEve(int const trackID);
```

Similar to FindTPCEve, but looks for the progenitor which exited the ECAL or TPC; this is intended for use with particles in the magnet yoke, i.e. the MuID.

```
bool IsForebearOf(simb::MCParticle* const forbear, simb::MCParticle* const afterbear) const;
```

Is MCParticle afterbear in the descent tree of MCParticle forbear? Does NOT use the adopted EveIdCalculator; just walks up the tree. MCParticle equality is tested by comparison of the MCParticle::TrackID() fields. Returns true if forbear == afterbear.

```
art::Ptr<simb::MCTruth> const
ParticleToMCTruth(simb::MCParticle* const p) const;
```

From an MCParticle to the MCTruth that it originates from.

*The following should work as long as there are also RawDigit and EnergyDeposit data products in the event, with an art::Assns between them* -----

*First we'll need to define HitsIDE:*

```
struct HitIDE {
    int trackID;           ///< Geant4 supplied trackID
    float energyFrac;      ///< fraction of CaloRawDigit energy from the particle with this trackID
    T LorentzVector position; ///< deposited energy-weighted mean true 4-position [cm, ns]
    T LorentzVector momentum; ///< deposited energy-weighted mean true 4-momentum [GeV/c, GeV]

    HitIDE() {}
    HitIDE(int id, float ef, float e, const T LorentzVector& pos, const T LorentzVector& mom, )
        : trackID(id), energyFrac(ef), energyTot(e), position(pos), momentum(mom) {}
    bool operator == (HitIDE const& hitr) const {...}
    bool operator < (HitIDE const& hitr) const {...}
};
```

The position and momentum quantities in HitIDE hold the energy deposit (i.e. the ionization) weighted position and momentum of the track. This is filled in ChannelToHitIDEs and is needed by GarAna. If SplitDEps is true (which is the default) in BackTracker.fcl, pad response functions Functions should be used to subdivide the energy deposit into different channels.

```
std::vector<HitIDE>
HitToHitIDEs(art::Ptr<rec::Hit> const& hit) const;
std::vector<HitIDE>
HitToHitIDEs(          rec::Hit const& hit) const;
```

This is the HitToTrackID method used in tpcpatreccheat\_module.cc, but we change the name for clarity. The returned cheat::HitIDE is a struct defined in BackTrackerCore.h and it includes the GEANT track ID, a fraction of the RawDigit energy from the particle with that track ID, and an "ionization" which seems to be charge in that channel but outside a certain time window.

```
std::vector<art::Ptr<rec::Hit>> const
ParticleToHits(simb::MCParticle* const p,
               std::vector<art::Ptr<rec::Hit>> const& allhits,
               checkNeutrals=false) const;
```

Returns all the TPC hits that have some energy from a particle. If you want to find the fraction of the energy in a hit from that particle, you need a HitIDE, found with HitToHitIDEs, above and this method only returns hits with that fraction over the value given by fMinHitEnergyFraction, which is specified in BackTracker.fcl; the default value of 0.1 is probably fine. I guess. If checkNeutrals==false, will return immediately for input particle is a photon, neutron, pi0 or either a mu or e neutrino thereby returning an empty vector promptly.

```
std::vector<HitIDE>
ChannelToHitIDEs(raw::Channel_t const& channel,
                 double const start, double const stop) const);
```

The above 3 public methods are wrappers to this private one. It works by looping through all the EnergyDeposits for the channel as given by fChannelToEDepCol, which in turn is filled by RebuildNoSC.

```
std::pair<double,double>
HitPurity(simb::MCParticle* const p,
          std::vector<art::Ptr<rec::Hit>> const& hits,
          bool weightByCharge=false) const;
```

Input is a single MCParticle and a collection of hits in the TPC; the return value is the fraction, and the binomial error on the fraction, of the hits that are from the MCParticle. Warning: the standard binomial error formula is dubious when used with weights.

```
std::pair<double,double>
HitEfficiency(simb::MCParticle* const p,
              std::vector<art::Ptr<rec::Hit>> const& hits,
              std::vector<art::Ptr<rec::Hit>> const& allhits,
              bool weightByCharge=false) const;
```

Input is a single MCParticle and two collections of hits in the TPC. The allhits collection should be all the hits in the event, or maybe all the plausible hits in the event. The intent is that allhits is all the hits that MCParticle created. The hits collection might be, say, all the hits in a reconstructed track from that MCParticle. The returned pair of doubles is the efficiency of track reconstruction and the binomial uncertainty. Again, the standard binomial error formula is dubious when used with weights.

*The following should work as long as there are also CaloRawDigit and CaloDeposit data products for the ECAL in the event, with an art::Assns between them* -----  
*"Calorimeter" means the combination of the ECAL and any MuID; the muon ID software is a 2nd instance of the ECAL art modules, run on a different geometry. The data for both are handled as one in the BackTracker.*

*First we'll need to define CalIDE:*

```
struct CalIDE {
    int trackID;           ///< Geant4 supplied trackID
    float energyFrac;      ///< fraction of CaloRawDigit energy from the particle with this trackID
    float energyTot        ///< total energy for this trackID. In units of probably-GeV.

    CalIDE() {}
    CalIDE(int id, float ef, float e)
        : trackID(id), energyFrac(ef) {}
};
```

*Second, create a typedef raw::CellID for long long int. It's in CaloRawDigit.h. Including this is a small but ugly addition to our dependencies. We should collect all these typedefs in one place. I've put that on the task list.*

```
std::vector<CalIDE>
CaloHitToCalIDEs(art::Ptr<rec::CaloHit> const& hit);
std::vector<CalIDE>
CaloHitToCalIDEs(          rec::CaloHit const& hit);
```

Similar to TPCHitToHitIDEs except that the CaloDeposit::TrackID should be the track ID of the particle coming into the calorimeter. That chase up the tree has to be done in CellIDToCalIDEs. There is reportedly a similar functionality in edepsim.

```
std::vector<art::Ptr<rec::CaloHit>> const
ParticleToCaloHits(simb::MCParticle* const p,
                   std::vector<art::Ptr<rec::CaloHit>> const& allhits) const;
```

Returns all the calorimeter hits that have some energy from TPC-leaving particle (see FindTPCEve). Input allhits is all plausible hits in the ECAL calorimeter that this method will search through.

```
std::vector<CalIDE>&
CellIDToCalIDEs(gar::rec::CaloHit);
```

The above 3 methods are wrappers to this private one. Works by looping through all the CaloDeposits for the CellID as given by fCellIDToCaloDepCol. Uses FindTPCEve.

```
std::pair<double,double>
CaloHitPurity(simb::MCParticle* const p,
              vector<art::Ptr<rec::CaloHit>> const& hits,
              bool weightByCharge=false) const;
```

Corresponding to HitCollectionPurity, except on the calorimeter side.

```
std::pair<double,double>
CaloHitEfficiency(simb::MCParticle* const p,
                  vector<art::Ptr<rec::CaloHit>> const& hits,
                  std::vector<art::Ptr<rec::CaloHit>> const& allhits,
                  bool weightByCharge=false) const;
```

Corresponding to HitCollectionEfficiency, except on the calorimeter side.

*The following should work as long as there are also Track data products in the event, with (presumably chained) art::Assns back to the Hits* -----  
*Again, "calorimeter" means the combination of the ECAL and any MuID; the muon ID software is a 2nd instance of the ECAL art modules, run on a different geometry. The data for both are handled as one in the BackTracker.*

```
std::vector<art::Ptr<rec::Hit>> const
TrackToHits(rec::Track* const t);
```

Uses the chained associations to get a collection of Hits for this Track.

```
std::vector<art::Ptr<rec::Hit>> const
TPCClusterToHits(rec::TPCCluster* const clust);
```

A wrapper to the unordered\_map which is private in BackTrackerCore and filled by BackTracker\_service.cc to get a collection of rec::Hit for this TPCCluster.

```
std::vector<art::Ptr<rec::TPCCluster>> const
TrackToTPCCluster(rec::Track* const t);
```

A wrapper to the unordered\_map which is private in BackTrackerCore and filled by BackTracker\_service.cc to get a collection of rec::Hit for this Track.

```
double TrackToTotalEnergy(rec::Track* const t)
```

Maybe not the best name - it actually gives the total ionization from HitIDEs.

```
std::vector<std::pair<simb::MCParticle*,float>>
TrackToMCParticles(rec::Track* const t);
```

Uses TrackToHits and then HitToHitIDEs to get energy fractions for each MCParticle that might match the track; and returns that list of candidate MCParticles sorted by their contribution. Electrons that have parents that are one of the other candidate MCParticles get their ionization included in with that parent. The floatin the returned vector is the fraction of the reconstructed track's energy attributed to each MCParticle.

```
vector<art::Ptr<rec::Track>>
MCParticleToTracks(simb::MCParticle* const p,
                  vector<art::Ptr<rec::Track>> const& tracks);
```

The inverse operation to TrackToMCParticles. Just loops over the input array of reconstructed tracks and see which tracks have more than fTrackFracMCP of their ionization energy from the input MCParticle. However, because of the merging of electrons in TrackToMCParticles, will return a zero-length vector if the argument is one of those electrons.

*The following should work as long as there are also Cluster data products in the event, with (presumably chained) art::Assns back to the CaloHits* -----  
*Again, "calorimeter" means the combination of the ECAL and any MuID; the muon ID software is a 2nd instance of the ECAL art modules, run on a different geometry. The data for both are handled as one in the BackTracker.*

```
std::vector<art::Ptr<rec::CaloHit>> const
ClusterToCaloHits(rec::Cluster* const c);
```

Uses the chained associations to get a collection of CaloHits for this Cluster.

```
std::vector<std::pair<simb::MCParticle*,float>>
ClusterToMCParticles(rec::Cluster* const c);
```

Uses ClusterToMCParticles and then CaloHitToCalIDEs to get energy fractions for each MCParticle that might match the cluster; and returns that list of candidate MCParticles sorted by their contribution. Gets the MCParticle entering the calorimeter using FindTPCEve.

```
vector<art::Ptr<rec::Cluster>>
MCParticleToClusters(simb::MCParticle* const p,
                     vector<art::Ptr<rec::Cluster>> const& clusters);
```

The inverse operation to ClusterToMCParticles.

```
bool ClusterCreatedMCParticle(simb::MCParticle* const p,
                              rec::Cluster* const c);
```

Uses MCPartsInCluster to get a collection of MCParticles in the cluster. Then uses IsForebearOf to see if MCParticle p is descended from any of those MCParticles. If p's ionization created c, then the answer is yes!

```
bool MCParticleCreatedCluster(simb::MCParticle* const p,
                              rec::Cluster* const c);
```

Uses MCPartsInCluster to get a collection of MCParticles in the cluster. Then uses IsForebearOf to see if any of those MCParticles are descended from MCParticle p. If p's ionization created c, then the answer is yes!

```
std::vector<simb::MCParticle*> MCPartsInCluster(rec::Cluster* const c);
```

A private method used by ClusterCreatedMCParticle and MCParticleCreatedCluster to use ClusterToCaloHits, CaloHitToCalIDEs and TrackIDToParticle to get a collection of MCParticles in the cluster.