Analysis TTreeFormat

From GlueXWiki

Contents

TTree Format - Overview

- Physics Analysis Root TTree (PART) format.

Data Hierarchy

- One TTree per DReaction, each stored in the ROOT files specified by the user.
  - e.g., If 2 DReactions: missing & detected recoil proton: 2 different trees, could be in separate files or the same file.
- One TTree entry per event.
- All particle data stored in arrays/TClonesArray's: one array index per particle.
  - Thrown particles
  - Reconstructed neutral and charged hypotheses (by default: only used ones, but can save all: DReaction setting)
  - Beam photons that are later used in combos (unused beam particles are NOT saved)
  - Combo particle information
- Event-independent information (e.g. the target, the DReaction decay chain, etc.) is stored in TTree::fUserInfo (a TList*)

DSelector

- Enables C++ interface to TTree data, provides PROOF-Lite launcher, and much more.
- Instructions for making and using a DSelector can be found at: Link

TTree Format: Simulated Data

Thrown Non-Particle Data

```
// EVENT DATA
"RunNumber": UInt_t
"EventNumber": ULong64_t
"MCWeight": Float_t

// # PARTICLES //array size of the thrown particle branches
"NumThrown": UInt_t

// THROWN REACTION INFO
"NumPIDThrown_FinalState": ULong64_t //the # of thrown final-state particles (+ pi0) of each type (multiplexed in base 10)
                                       //types (in order from 10^0 -> 10^15): g, e+, e-, nu, mu+, mu-, pi0, pi+, pi-, KLong, K+, K-, n, p,
                                       //e.g. particles decaying from final-state particles are NOT included (e.g. photons from pi0, muons
                                     //is sum of #-of-PID * 10^ParticleMultiplexPower() (defined in libraries/include/particleType.h)
                                     //ParticleMultiplexPower() returns a different power of 10 for each final-state PID type.
                                     //A value of 9 should be interpreted as >= 9.
"PIDThrown_Decaying": ULong64_t //the types of the thrown decaying particles in the event (multiplexed in base 2)
                                //not the quantity of each, just whether or not they were present (1 or 0)
                                //binary power of a PID is given by ParticleMultiplexPower() (defined in libraries/include/particleType.h)
                                //types: most Particle_t's that aren't final state (e.g. lambda, eta, phi, rho0, etc.) see ParticleMultipl
```

Thrown Beam Particle

- All branch names are prefixed with "ThrownBeam__"

```
//IDENTIFIER
"PID": Int_t //PDG ID value

//KINEMATICS: //At the production vertex
"X4": TLorentzVector //This is the TAGGED energy //Use THIS for binning your results //Is ZERO if NOT TAGGED
"P4": TLorentzVector
"GeneratedEnergy": Float_t
```

Thrown Products

- All branch names are prefixed with "Thrown__"
- NOTE: The only contains particles corresponding to the "FinalState" and "Decaying" tags of DMCThrown.
  - In other words: No resonances, no decay products of final-state particles, and no orphan particles.

```
//IDENTIFIERS / MATCHING
"ParentIndex": Int_t["NumThrown"] //the thrown particle array index of the particle this particle decayed from (-1 if none (e.g. photoproc
"PID": Int_t["NumThrown"] //PDG ID value

//MATCHING //only present if reconstructed data present (i.e. not if thrown-only tree)
"MatchID": Int_t["NumThrown"] //the "NeutralID"/"TrackID" of the reconstructed neutral/track that it is matched with (-1 for no match)
"MatchFOM": Float_t["NumThrown"] //Neutrals: confidence level //Tracks: #-matched-hits * hit_fraction //(-1 for no match)

//KINEMATICS: //Reported at the particle's production vertex
"X4": TClonesArray(TLorentzVector["NumThrown"])
"P4": TClonesArray(TLorentzVector["NumThrown"])
```

TTree Format: Combo-Independent Data

Non-Particle Data

```
// EVENT DATA
"RunNumber": UInt_t
"EventNumber": ULong64_t
"L1TriggerBits": UInt_t

// PRODUCTION SPACETIME
"X4_Production": TLorentzVector //V3 from DVertex (kinfit), t from RF (propagated to V3)

// # PARTICLES //these are the array sizes for the particle branches
"NumBeam": UInt_t
"NumChargedHypos": UInt_t
"NumNeutralHypos": UInt_t

// TOPOLOGY //only present if simulated data
"IsThrownTopology": Bool_t //Does the DReaction decay chain match the thrown decay chain

// UNUSED TRACKS
"NumUnusedTracks": UChar_t

//NUM COMBOS
"NumCombos": UInt_t //size of all of the particle-combo-content arrays
```

Beam Particles (If Used in Combo)

- Only the beam particles that are included in at least one combo are present.
- All branch names are prefixed with "Beam__"

```
//ONLY PRESENT IF BEAM USED IN PARTICLE COMBOS

//IDENTIFIERS / MATCHING
"PID": Int_t["NumBeam"] //PDG ID value
"IsGenerator": Bool_t["NumBeam"] // kTRUE/kFALSE if matches the generator beam photon (-1 for no match) //only present if simulated data

//KINEMATICS: MEASURED //At the production vertex
"X4_Measured": TClonesArray(TLorentzVector["NumBeam"]) //position is at the production vertex (same as X4_Production(), except the time)
"P4_Measured": TClonesArray(TLorentzVector["NumBeam"])
```

Charged Track Hypotheses

- Includes all hypotheses, whether they appear in the combos or not.
- All branch names are prefixed with "ChargedHypo__"

```
//IDENTIFIERS / MATCHING
"TrackID": Int_t["NumChargedHypos"] //each physical particle has its own # (to keep track of different pid hypotheses for the same particl
"PID": Int_t["NumChargedHypos"] //PDG ID value
"ThrownIndex": Int_t["NumChargedHypos"] //the array index of the thrown particle it is matched with (-1 for no match) //only present if si

//KINEMATICS: MEASURED  //At the production vertex
"P4_Measured": TClonesArray(TLorentzVector["NumChargedHypos"])
"X4_Measured": TClonesArray(TLorentzVector["NumChargedHypos"]) //t is the measured value in TOF/BCAL/FCAL projected back to Position_Measu

//TRACKING INFO:
"NDF_Tracking": UInt_t["NumChargedHypos"]
"ChiSq_Tracking": Float_t["NumChargedHypos"]
"NDF_DCdEdx": UInt_t["NumChargedHypos"]
"ChiSq_DCdEdx": Float_t["NumChargedHypos"]
"dEdx_CDC": Float_t["NumChargedHypos"]
"dEdx_FDC": Float_t["NumChargedHypos"]

//TIMING INFO
"HitTime": Float_t["NumChargedHypos"] //the system that is hit is in order of preference: BCAL/TOF/FCAL/ST
                                      //to determine which, look whether energy was deposited in these systems
"RFDeltaTVar": Float_t["NumChargedHypos"] //Variance of X4_Measured.T() - RFTime, regardless of which RF bunch is chosen.
                                          //Can be used to compute timing ChiSq //RF bunch is combo-dependent

//PID INFO
"Beta_Timing": Float_t["NumChargedHypos"] // = Path_Length/(c*Delta_t)
"ChiSq_Timing": Float_t["NumChargedHypos"]
"NDF_Timing": UInt_t["NumChargedHypos"]

//HIT ENERGY:
"dEdx_TOF": Float_t["NumChargedHypos"]
"dEdx_ST": Float_t["NumChargedHypos"]
"Energy_BCAL": Float_t["NumChargedHypos"]
"Energy_BCALPreshower": Float_t["NumChargedHypos"]
"Energy_FCAL": Float_t["NumChargedHypos"]

//SHOWER WIDTH:
```

```
"SigLong_BCAL" Float_t["NumChargedHypos"] // Longitudinal (outward radially from the target) shower width
"SigTheta_BCAL" Float_t["NumChargedHypos"] // Theta shower width
"SigTrans_BCAL" Float_t["NumChargedHypos"] // Transverse (azimuthal) shower width

//SHOWER MATCHING:
"TrackBCAL_DeltaPhi": Float_t["NumChargedHypos"] //999.0 if not matched //units are radians
"TrackBCAL_DeltaZ": Float_t["NumChargedHypos"] //999.0 if not matched //Track position - BCAL Shower
"TrackFCAL_DOCA": Float_t["NumChargedHypos"] //999.0 if not matched
```

Neutral Particle Hypotheses

- All branch names are prefixed with "NeutralHypo__"
- Includes all hypotheses, whether they appear in the combos or not.
- Discussion on P4 & X4:
  - Note that P4 is not present because it is defined by X4, and X4 is not present because it is defined by the tracks, which are combo-dependent
  - For combo particles, P4 & X4 are listed for each combo
  - If not used in a combo, can be computed using the shower hit information and the vertex & RF-time of your choosing (e.g. combo production-vertex, RF-time)
- To determine whether is BCAL or FCAL, see which system has non-zero energy

```
//IDENTIFIERS / MATCHING
"NeutralID": Int_t["NumNeutralHypos"] //each physical particle has its own # (to keep track of different pid hypotheses for the same parti
"PID": Int_t["NumNeutralHypos"] //PDG ID value
"ThrownIndex": Int_t["NumNeutralHypos"] //the array index of the thrown particle it is matched with (-1 for no match) //only present if si

//KINEMATICS: MEASURED  //At the production vertex
"P4_Measured": TClonesArray(TLorentzVector["NumNeutralHypos"])
"X4_Measured": TClonesArray(TLorentzVector["NumNeutralHypos"]) //t is the measured value in TOF/BCAL/FCAL projected back to Position_Measu

//MEASURED PID INFO
"Beta_Timing": Float_t["NumNeutralHypos"] // = Path_Length/(c*Delta_t)
"ChiSq_Timing": Float_t["NumNeutralHypos"] //-1 if not photon
"NDF_Timing": UInt_t["NumNeutralHypos"] //0 if not photon

//SHOWER INFO
"X4_Shower": Float_t["NumNeutralHypos"] //location/time of the reconstructed shower
"Energy_BCAL": Float_t["NumNeutralHypos"] //is 0.0 if shower in FCAL
"Energy_BCALPreshower": Float_t["NumNeutralHypos"] //is 0.0 if shower in FCAL
"Energy_FCAL": Float_t["NumNeutralHypos"] //is 0.0 if shower in BCAL

//SHOWER WIDTH:
"SigLong_BCAL" Float_t["NumNeutralHypos"] // Longitudinal (outward radially from the target) shower width
"SigTheta_BCAL" Float_t["NumNeutralHypos"] // Theta shower width
"SigTrans_BCAL" Float_t["NumNeutralHypos"] // Transverse (azimuthal) shower width

//NEARBY TRACKS
"TrackBCAL_DeltaPhi": Float_t["NumNeutralHypos"] //is delta to nearest track, is 999.0 if no tracks on BCAL
"TrackBCAL_DeltaZ": Float_t["NumNeutralHypos"] //is delta to nearest track, is 999.0 if no tracks on BCAL
"TrackFCAL_DOCA": Float_t["NumNeutralHypos"] //is DOCA to nearest track, is 999.0 if no tracks on FCAL

//PHOTON PID INFO
   //Computed using DVertex (best estimate of reaction vertex using all "good" tracks)
   //Can be used to compute timing chisq //is invalid (0) for non-photons
"PhotonRFDeltaTVar": Float_t["NumNeutralHypos"] //Variance of DVertexX4.T() - RFTime, regardless of which RF bunch is chosen. //RF bunch i
```

TTree Format: Combo-Dependent Data

- All particle combo data is stored in arrays: array entries correspond to different particle combos

Particle-Independent Data

```
//CUT FLAG
"IsComboCut": Bool_t["NumCombos"] //if true, combo has been previously cut (all kFALSE originally, user can apply cuts in TSelector, chang

//COMBO THROWN MATCHING //not present if not simulated data
"IsTrueCombo": Bool_t["NumCombos"] //"IsThrownTopology" = kTRUE, each particle has the right PID, and the combo particle chain matches the
"IsBDTSignalCombo": Bool_t["NumCombos"] //Similar to "IsTrueCombo", except other thrown topologies that decay to the DReaction topology ar
                                        //Note that if you have an &omega; or &phi; in your DReaction, you still have to filter your combo
                                        //input to remove duplicate entries. This is because the omega & phi masses are not constrained in
```

```
                                        //nor should they be in the BDT, so you have duplicate entries from the point-of-view of the BDT c
                                        //(e.g. which pions decayed from the omega, and which ones didn't, are irrelevant to the BDT).

//RF
"RFTime_Measured": Float_t["NumCombos"] //reported at center of target
"RFTime_KinFit": Float_t["NumCombos"] //reported at center of target //only if spacetime kinematic fit performed

//KINEMATIC FIT
"ChiSq_KinFit": Float_t["NumCombos"] //only if kinematic fit performed
"NDF_KinFit": UInt_t["NumCombos"] //only if kinematic fit performed // = 0 if kinematic fit doesn't converge

//UNUSED ENERGY
"Energy_UnusedShowers": Float_t["NumCombos"] // summed energy of neutral showers in the event not included in the combo (requiring unused

//UNUSED TRACKS //For tracks unused by combo, the hypo chosen is the one with the best tracking FOM
"SumPMag_UnusedTracks": Float_t["NumCombos"]
"SumP3_UnusedTracks": TClonesArray(TVector3["NumCombos"])
```

Particle Branch-Name Prefixes

Example Reaction (b1pi):

- γ p $\rightarrow\omega$, π^+, π^-, (p)
  - $\omega \rightarrow \pi^+, \pi^-, \pi^0$
    - $\pi^0 \rightarrow \gamma\,\gamma$

Branch Names:

- Beam: "ComboBeam"
- Detected: "PiMinus1", "PiPlus1", "PiPlus2", "PiMinus2", "Photon1", "Photon2"
- Decaying: "DecayingPi0"
- Missing: "MissingProton"

Combo Beam Particles (If Any)

- All branch names are prefixed with "ComboBeam__"
  - E.g. "ComboBeam__BeamIndex"

```
//IDENTIFIER
"BeamIndex": Int_t["NumCombos"] //array index to the "Beam__" branches that correspond to this particle

//KINEMATICS: KINFIT //At the interaction vertex //only present if kinfit performed
"X4_KinFit": TClonesArray(TLorentzVector["NumCombos"]) //not present if p4-only fit
"P4_KinFit": TClonesArray(TLorentzVector["NumCombos"]) //not present if vertex-only or spacetime-only fit, unless beam is charged
```

Combo Tracks (If Any)

- All branch names are prefixed with the particle name
  - E.g. "Proton__ChargedIndex", "PiMinus1__P4_KinFit"

```
//IDENTIFIER
"ChargedIndex": Int_t["NumCombos"] //array index to the "ChargedHypo__" branches that correspond to this particle

//PID INFO: MEASURED //using combo RF bunch
"Beta_Timing_Measured": Float_t["NumCombos"] // = Path_Length/(c*Delta_t)
"ChiSq_Timing_Measured": Float_t["NumCombos"]

//PID INFO: KINFIT //using combo RF bunch //not present if time constrained //uses combo vertex & p4 if kinfit
"Beta_Timing_KinFit": Float_t["NumCombos"] // = Path_Length/(c*Delta_t)
"ChiSq_Timing_KinFit": Float_t["NumCombos"]

//KINEMATIC FIT KINEMATICS //only present if kinfit performed
"X4_KinFit": TClonesArray(TLorentzVector["NumCombos"]) //not present if p4-only fit
"P4_KinFit": TClonesArray(TLorentzVector["NumCombos"])
```

Combo Neutrals (If Any)

- All branch names are prefixed with the particle name
  - E.g. "Photon1__NeutralIndex", "Neutron__P4_KinFit"

```
//IDENTIFIER
"NeutralIndex": Int_t["NumCombos"] //array index to the "NeutralHypo__" branches that correspond to this particle
                                  //Note that they may not have the same PID (and thus P4) as this!!
                                     //If this is a PID not created by default (e.g. K0Long)

//KINEMATICS: MEASURED  //At the production vertex
"P4_Measured": TClonesArray(TLorentzVector["NumCombos"])
"X4_Measured": TClonesArray(TLorentzVector["NumCombos"]) //t is the measured value in TOF/BCAL/FCAL projected back to Position_Measured

//MEASURED PID INFO
"Beta_Timing_Measured": Float_t["NumCombos"] // = Path_Length/(c*Delta_t)
"ChiSq_Timing_Measured": Float_t["NumCombos"] //only present if photon

//KINEMATIC FIT PID INFO
"Beta_Timing_KinFit": Float_t["NumCombos"] // = Path_Length/(c*Delta_t) //not present if p4-only fit
"ChiSq_Timing_KinFit": Float_t["NumCombos"] //only present if photon //not present if p4-only fit

//KINEMATIC FIT KINEMATICS //only present if kinfit performed
"X4_KinFit": TClonesArray(TLorentzVector["NumCombos"]) //not present if p4-only fit
"P4_KinFit": TClonesArray(TLorentzVector["NumCombos"])
```

Combo Decaying Particles (If Any, If Detached/KinFit)

- All branch names are prefixed with "Decaying" and the particle name
  - E.g.: "DecayingPi0__X4"

```
//KINEMATICS: //At the decay vertex
"X4": TLorentzVector["NumCombos"] //only present if has a detached vertex //kinematic fit result if kinfit performed, else reconstructed f
"PathLengthSigma": Float_t["NumCombos"] //only present if has a detached vertex and both vertices are fit
"P4_KinFit": TLorentzVector["NumCombos"] //only present if kinfit performed
```

Combo Missing Particles (If Any & If KinFit)

- All branch names are prefixed with "Missing" and the particle name
  - E.g.: "MissingProton__P4_KinFit"

```
//KINFIT KINEMATICS: //At its production vertex //only present if kinfit performed
"P4_KinFit": TLorentzVector["NumCombos"]
```

TTree Format: DReaction Info

- Stored in TTree::fUserInfo (a TList*)

- "ParticleNameList": TList of the names of the reaction particles in the tree, in the order they were specified in the DReaction.

- "MiscInfoMap": TMap of TObjString -> TObjString
  - "KinFitType" -> DKinFitType (converted to TObjString)
  - "Target__PID" -> int (converted to TObjString): PDG PID of target particle //if a target particle was specified
  - "Target__Mass" -> double (converted to TObjString): Mass of the target particle. //if a target particle was specified
  - "Missing__PID" -> int (converted to TObjString): PDG PID of missing particle //if a missing particle was specified
  - "Target__CenterX" -> double (converted to TObjString): x-coordinate of target center
  - "Target__CenterY" -> double (converted to TObjString): y-coordinate of target center
  - "Target__CenterZ" -> double (converted to TObjString): z-coordinate of target center
  - "MissingNAME__Mass" -> double (converted to TObjString): Mass of the 'NAME' missing particle (e.g. 'NAME' = Proton). //if a missing particle was specified
  - "DecayingNAME__Mass" -> double (converted to TObjString): Mass of the 'NAME' decaying particle (e.g. 'NAME' = Pi0). //if decaying particles were present

- "NameToPIDMap": TMap of "UniqueParticleName" (TObjString) -> int (PDG) (converted to TObjString)

- "NameToPositionMap": TMap of "UniqueParticleName" (TObjString) -> "StepIndex_ParticleIndex" (stored in TObjString) (ParticleIndex = -1 for initial, -2 for target, 0+ for final state)
- "PositionToNameMap": TMap of "StepIndex_ParticleIndex" (stored in TObjString) (ParticleIndex = -1 for initial, -2 for target, 0+ for final state) -> "UniqueParticleName" (TObjString)
- "PositionToPIDMap": TMap of "StepIndex_ParticleIndex" (stored in TObjString) (ParticleIndex = -1 for initial, -2 for target, 0+ for final state) -> int (PDG) (converted to TObjString)
- "DecayProductMap": TMap of "DecayingParticleName" (TObjString) -> "DecayProductNames" (stored in a TList of TObjString objects). Excludes resonances and intermediate decays (e.g. if $\Xi^- \to \pi^- \Lambda \to \pi^- \pi^- p$: will be $\Xi^- \to \pi^- \pi^- p$ and Λ decay not listed)

Usage

Create TTrees

- To save data to a TTree for a given DReaction, TTree output must be first be enabled for that reaction. See DReaction Control Variables (https://halldweb.jlab.org/wiki/index.php/Analysis_DReaction#DReaction_Control_Variables) for details.
  - Note: Only one thrown tree will be created during program execution. If the DEventWriterROOT::Create_ThrownTree() function is called more than once, nothing happens on subsequent calls.

```
#include "ANALYSIS/DEventWriterROOT.h"
//In plugin brun():
const DEventWriterROOT* locEventWriterROOT = NULL;
locEventLoop->GetSingle(locEventWriterROOT);
locEventWriterROOT->Create_DataTrees(locEventLoop); //creates TTrees for all output-enabled DReactions
locEventWriterROOT->Create_ThrownTree("tree_b1pi_thrownmc.root"); //optional: create a ttree containing only the thrown data //string is c
```

Save Data to TTree

- The below only saves the particle combinations (for TTree-output-enabled DReaction's created in the factory specified by the tag) that survived all of the DAnalysisAction cuts.

```
//In plugin evnt()
const DEventWriterROOT* locEventWriterROOT = NULL;
locEventLoop->GetSingle(locEventWriterROOT);
locEventWriterROOT->Fill_DataTrees(locEventLoop, "b1pi_hists"); //string is the DReaction factory tag that the DReactions were created in
```

- The below allows you to choose which DParticleCombo's (locParticleCombos) of which DReaction's (locReaction) to save.
  - Beware: the locParticleCombos MUST have originated from the locReaction or else this will probably crash (can check DParticleCombo::Get_Reaction()).

```
//In plugin evnt()
#include "ANALYSIS/DEventWriterROOT.h"
vector<const DEventWriterROOT*> locEventWriterROOTVector;
locEventLoop->Get(locEventWriterROOTVector); //creates the TTrees for all DReactions upon first call
locEventWriterROOTVector[0]->Fill_Tree(locEventLoop, locReaction, locParticleCombos);
```

- The below fills a TTree that only contains the thrown particle data.

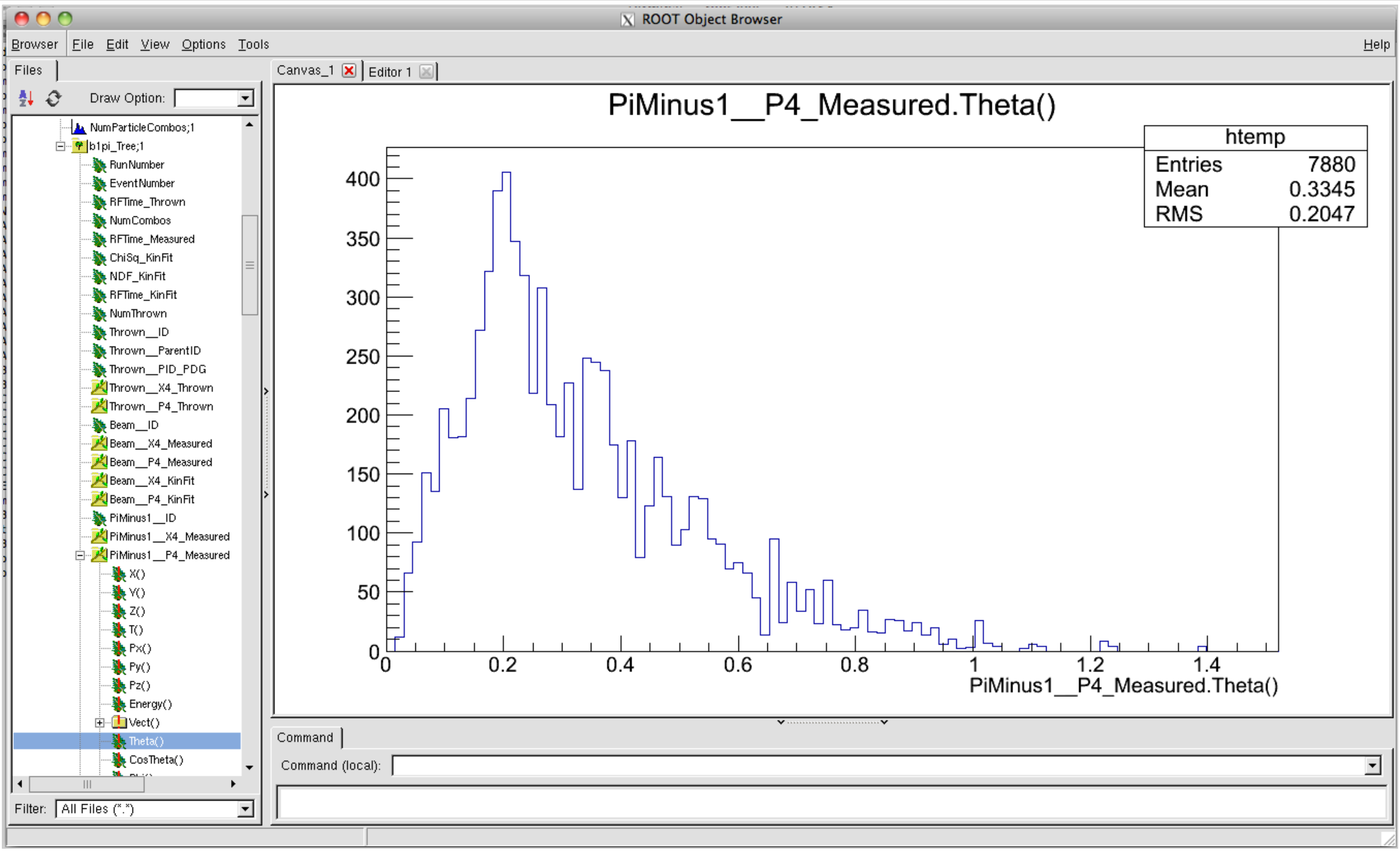
```
//In plugin evnt()
const DEventWriterROOT* locEventWriterROOT = NULL;
locEventLoop->GetSingle(locEventWriterROOT);
locEventWriterROOT->Fill_ThrownTree(locEventLoop);
```

Accessing TTree Data

- TTree:

```
MyTree->Draw("PiMinus1__P4_Measured->Theta()"); //draws all particle combinations
```

- TBrowser (draws all particle combinations):



b1pi Events

TSelector / TPROOF Links

- Documentation Link: PROOF (https://root.cern.ch/drupal/content/proof)
- Documentation Link: PROOF-Lite (https://root.cern.ch/drupal/content/proof-multicore-desktop-laptop-proof-lite)
- Documentation Link: TSelector (https://root.cern.ch/drupal/content/developing-tselector)
- Documentation Link: Full TSelector Example (with PROOF-Lite) (https://root.cern.ch/drupal/content/processing-proof)
- Documentation Link: Process Examples (https://root.cern.ch/drupal/content/basic-processing)
- Documentation Link: Large Output Files (https://root.cern.ch/drupal/content/handling-large-outputs-root-files)
- Documentation Link: Loading a macro for PROOF (https://root.cern.ch/drupal/content/loading-macro-or-class)
- Documentation Link: Working with packages (https://root.cern.ch/drupal/content/working-packages-par-files)

Usage - Advanced

Custom Branches

- You can create and fill custom branches by inheriting from the DEventWriterROOT class to create your own writer class.
- Use the trunk/scripts/analysis/MakeEventWriterROOT.pl script to generate the necessary code to do this.
- Run this perl script with no arguments to get complete usage instructions.

Preventing Double-Counting

- Since you can have multiple particle combinations per event, you have to be very careful to make sure you aren't double-counting when filling your histograms.
  - For example, if you're histogramming the invariant mass of the π^0's decay to $\gamma\gamma$ in b1pi events using the measured photon data, multiple combinations may use the same showers for the photons, while having different tracks for the other particles.

Converting for AmpTools

- To convert the TTree for use as input to AmpTools, use the tree_to_amptools in the gluex_root_analysis repository. Run with no arguments for instructions.