

Standard Ntuple (NtpSt/SR/MC/TH) Tutorial

A Brief introduction to the Standard Ntuples

- What are they?
- Accessing the files
- Structure and content
- How to use them
- References

Standard Ntuples (NtpSt/SR/MC/TH), What are they?

- Standard Ntuples are those ntuples produced during production running as an end product of Standard Reconstruction.
 - Ntuple data is contained in files ending with .sntp.R<release>.root (full ntuples) and .snts.R<release>.root (slim ntuples).
 - Recent releases used in production have been R1.12,1.14,1.16 (gearing up).
 - There are also Cambridge ntuples (produced as the end product of Cambridge Reconstruction), and Analysis Ntuples, deriving from Standard ntuples.
- Each Standard Ntuple file contains one or more ntuples, each in the form of a ROOT TTree.
- The ntuple trees produced during production (up until R1.14) were:
 - NtpSR containing a summary of the standard reconstruction results.
 - NtpMC containing a summary of mc truth information.
 - NtpTH containing “truth helper” information used to match reconstruction results with the mc truth information.
- These 3 ntuples have been replaced in production running with a single ntuple TTree (as of R1.14)
 - NtpSt contains all data previously stored in the 3 separate ntuples.
 - The 3 separate ntuples are maintained and although not produced in production running, can still be produced in user’s individual jobs.

Standard Ntuples (NtpSt/SR/MC/TH), Accessing the files

- Instructions on how to find the ntuple files are given on the Getting data and Production sites off the Offline S/W web site: http://www-numi.fnal.gov/offline_software/srt_public_context/WebDocs/WebDocs.html

Links for Beginners - primers and cribs	WebDocs Navigation , Tutorials , Package Overviews , Concept Overviews , Setup base release at FNAL .
Documents - manuals, glossaries & FAQs	User Manual at FNAL ([HTML] , [PS]) Package , software and other glossaries. Minos software FAQ pages and old docs
Organisation - working and communicating	Global and local organization. Mailing list with archives (also older ones). HyperNews for discussions and meetings .
Tools - to develop, build & debug	Accessing Our Respository . CVS and SRT . Managing base and test releases Random help and tool reviews , utility scripts and hints on handling bugs .
Off-line Framework - for reconstruction & analysis	Packages , requirements ([HTML] [PS]), code browsers: doxygen [at Ox. , at FNAL , or local].
Reconstruction Software - for event analysis	Packages , De-multiplexers , Calibration Framework , Standard Reconstruction (SR) classes ([HTML] , [PS]), and to-do list .
Installation - of framework & support libs	Supported platforms , Software , Live CD and Database distribution
Applications - configuring and running	Preparation for and configuring of the offline job. Documentation on libMidad . How to run rotorooter , minfast and reeroot .
Code Development - both private and public	How to create and use a test release . Our coding conventions in full , hot list , emacs macros . Understanding errors messages . How to document your code /for doxygen. Debugging and optimising . Database Portability Issues .
Status - problems & progress	Code: Bug tracking through files and JIRA , todo list , changelog , List of Frozen Releases . Sites: CalDet status , FNAL Farm Production Statistics . Database: Nick's Current Problems Dennis' Minos-Oracle Status Reports/Daybook
Data - from Detectors & Databases	Data: Getting data , Run search , SAM Data Browsing , Data Handling , File Names , Standard N-tuples Database: Overview , Table Status , Distribution , Update log , Size , Web interface , Hardware DB development . Batch System: Using the FNALU batch system , Statistics Online & DAQ: CRL , Far/Near DAQ logs , FarDet Status Beam: GNUMI

Farm Production Site

- Information about latest production run, available files and where to get them

Getting data Site

- Information about location of files and how to get them via AFS (slim ntuples) or FNAL mass storage (full ntuples).

Standard N-tuples Site

- Introduction to ntuples, how to get them, example scripts to access data, references to more information.

Standard Ntuples (NtpSt/SR/MC/TH), Accessing the files

home Bookmarks WebMail Contact People Yellow Pages Download Find Sites Channels

MINOS MC Farm Production Statistics

These MC version R1.14 statistics were last updated at 14:37 on Monday June 13, 2005.

For other MC processing see [R0.8.0](#), [R1.6.1](#), [R1.7](#), [R1.9](#), [R1.12](#), or [R1.16](#).

Click for [FAR](#) or [NEAR](#) production processing.

Click a run type to view a list of runs processed with R1.14 and a link to monitor files.

Type	Pass	Fail	Cand GB	SNtp GB	SNtS GB
far	737	50	193.12	30.95	5.46
near	866	41	402.50	24.68	6.79
Total	1603	91	595.6	55.6	12.3

To date, 753.6 GB has been written to all R1.14 MC output streams.

The 12.3 GB of snts output data can also be found in Fermilab AFS space at [/afs/fnal.gov/files/data/minos/d10/recodata01](#) thru [recodata16](#)

Indexes to the afs files can be found at [/afs/fnal.gov/files/data/minos/d10/indexes](#).

The output files are stored on the Fermilab [ENSTORE](#) system and can be accessed from the MINOS cluster using `encp/dccp` or from elsewhere using `dcache`.

Data is stored in directories `/bnfs/minos/mc/out data/R1.14/near/far/nmock/fmock/cand data .../snto data`.

Files produced in farm production:

.cand.R1.XX.root files

- Full reconstruction, MC truth, and raw data.

.sntp.R1.XX.root files

- Full ntuple data, derived from full reconstruction and mc data.

.snts.R1.XX.root files

- Slim ntuple data. Same information as full ntuple, but all strip data has been removed.
- Available on afs space.

S. Kasahara

Standard Ntuples (NtpSt/SR/MC/TH), Structure and Content

The content of NtpSt ntuple is defined by data members of NtpStRecord class.

(\$SRT_PUBLIC_CONTEXT/StandardNtuple/NtpStRecord.h):

```

public:
// Ntuple is treated like a C-struct with public data members and
// rule-breaking field data members not prefaced by "f" and all
// lowercase, by popular demand.

SR summary {
NtpSREventSummary evthdr; // sr summary data
NtpSRShieldSummary vetohdr; // veto shield summary data
NtpSRCosmicRay crhdr; // cr data, filled from last recons. trk vertex
NtpSRDmxStatus dmxstatus; // status of demux
NtpSRDetStatus detstatus; // status of detector

MC summary {
NtpMCSummary mchdr; // mc summary data
NtpMCPhotonResult photon; // summary data from photon transport mc
NtpMCDetSimResult detsim; // summary data from det sim

// Standard reconstruction arrays
Arrays of SR objects {
TClonesArray* vetostp; //-> array of shield strips of type NtpSRShieldStrip
TClonesArray* stp; //-> array of strips of type NtpSRStrip
TClonesArray* slc; //-> array of slices of type NtpSRSlice
TClonesArray* clu; //-> array of showers of type NtpSRCluster
TClonesArray* shw; //-> array of showers of type NtpSRShower
TClonesArray* trk; //-> array of tracks of type NtpSRTrack
TClonesArray* evt; //-> array of events of type NtpSREvent

Arrays of MC objects {
// MC truth arrays
TClonesArray* mc; //-> array of NtpMCTruth
TClonesArray* stdhep; //-> array of NtpMCStdHep
TClonesArray* digihit; //-> array of NtpMCDigiScintHit

Arrays of TH objects {
// Truth helper arrays
TClonesArray* thstp; //-> array of NtpTHStrip
TClonesArray* thslc; //-> array of NtpTHSlice
TClonesArray* thshw; //-> array of NtpTHShower
TClonesArray* thtrk; //-> array of NtpTHTrack
TClonesArray* thevt; //-> array of NtpTHEvent

ClassDef(NtpStRecord,6)
};
    
```

The NtpSt ntuple (TTree) is created by:

- Filling one NtpStRecord object per Snarl, from data derived from the Raw/Cand/SimSnarlRecord.
- Persisting the filled NtpStRecord to the NtpSt TTree.
1 NtpStRecord ↔ 1 tree entry
- The NtpSt tree is structured to split the storage of object data members onto separate branches.
 - NtpStRecord in the tree main branch.
 - Each data member of the NtpStRecord becomes a branch on the NtpSt Ttree, e.g.:
NtpStRecord.crhdr
 - Further splitting of objects is done, down to basic types as necessary. Each split results in a subbranch. The last split to a basic data type is a “leaf” on the tree, e.g.:
NtpStRecord.crhdr.zenith

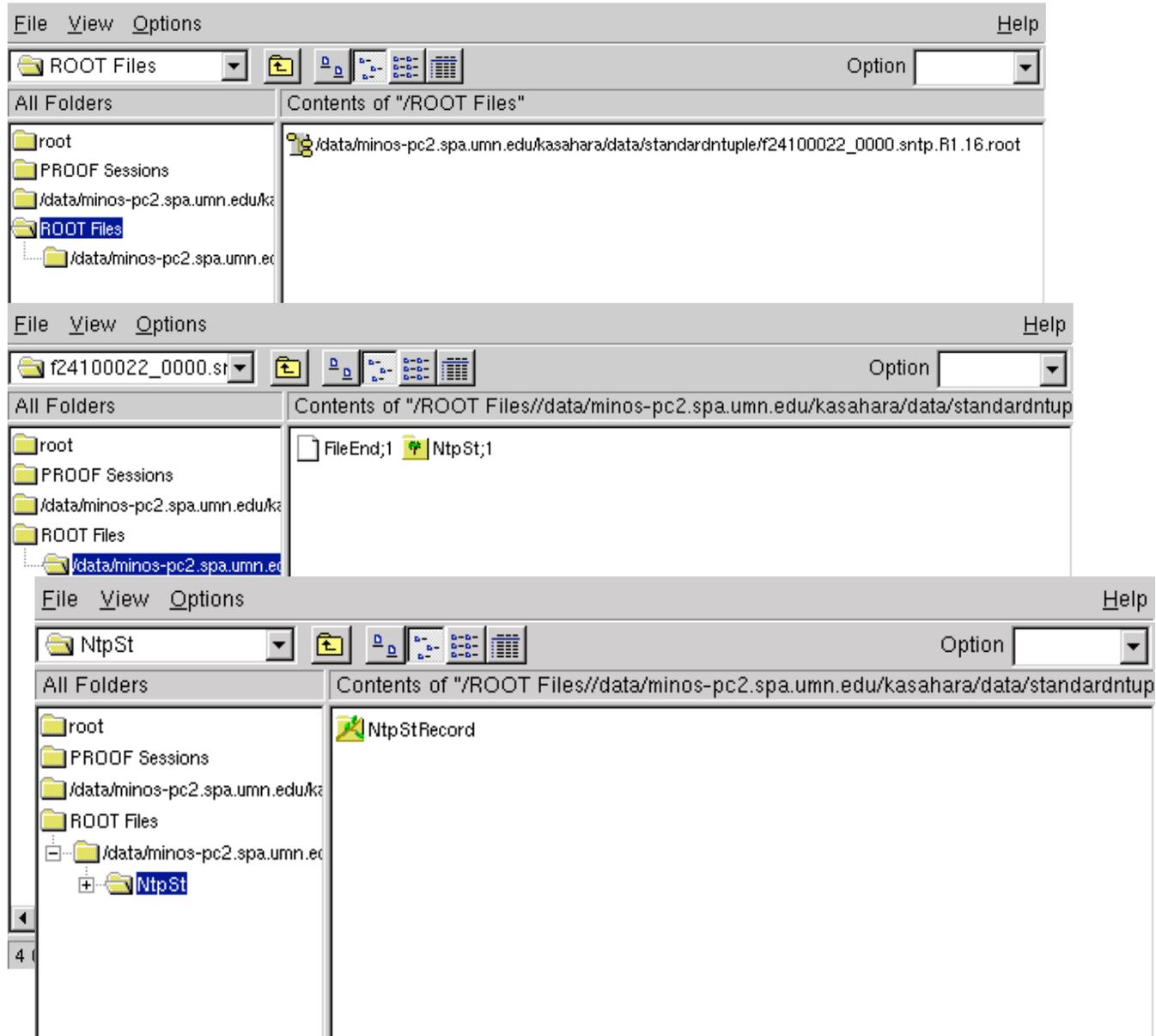
Standard Ntuples (NtpSt/SR/MC/TH), Structure and Content

Another way to view the NtpSt TTree structure is through a ROOT TBrowser:

```
[275]schubert@minos-pc2> root
*****
*          W E L C O M E  t o  R O O T          *
*          *          *          *          *
*   Version   4.03/05         4 June 2005      *
*          *          *          *          *
* You are welcome to visit our Web site      *
*   http://root.cern.ch                    *
*          *          *          *          *
*****

FreeType Engine v2.1.3 used to render TrueType fonts.
Compiled for linux with thread support.

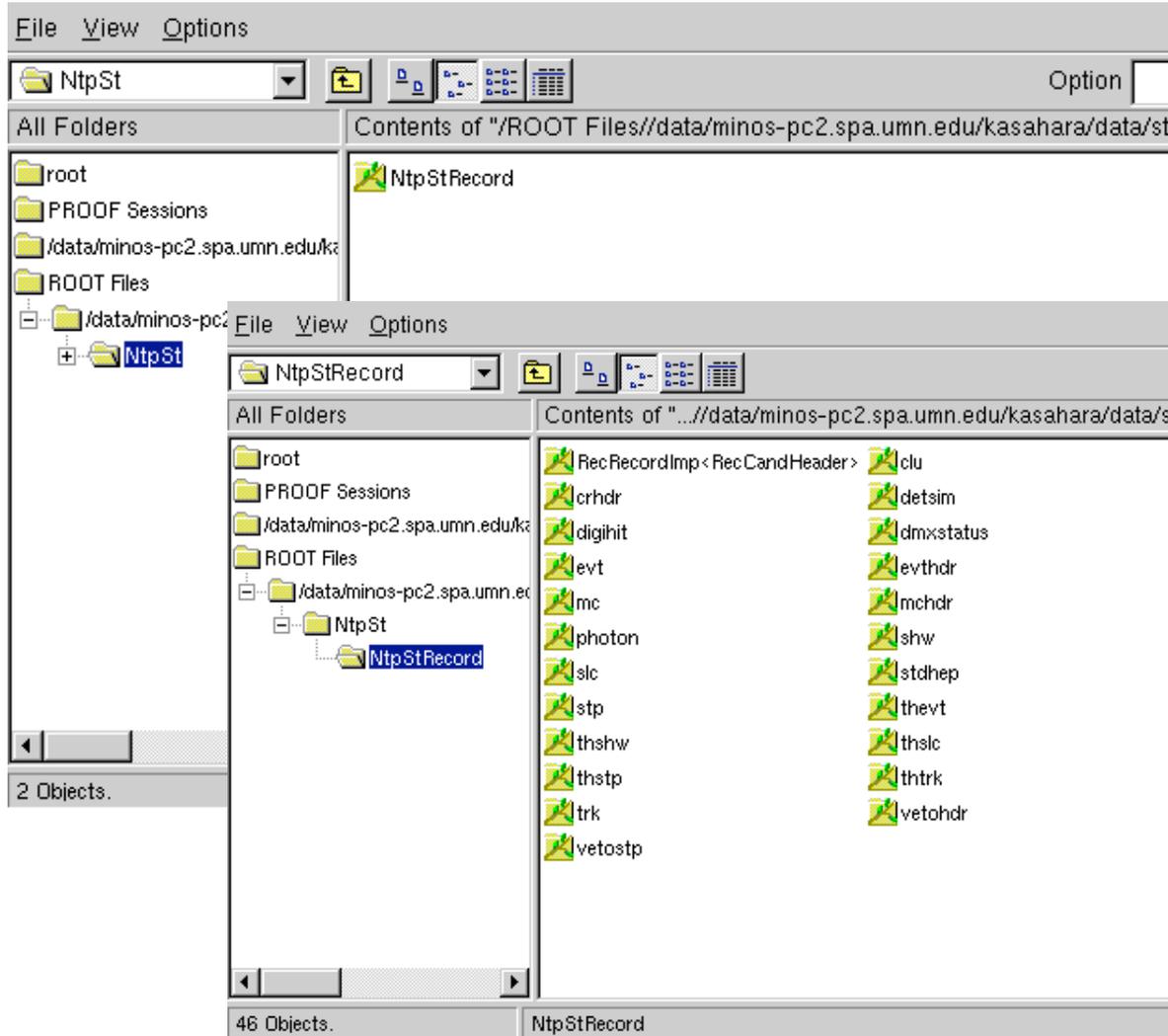
CINT/ROOT C/C++ Interpreter version 5.15.169, Mar 14 2005
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] b = new TBrowser();
root [1] █
```



June 14, 2005

Standard Ntuples (NtpSt/SR/MC/TH), Structure and Content

Another way to view the same thing is through a ROOT TBrowser:



```
public:
// Ntuple is treated like a C-struct with public data members and
// rule-breaking field data members not prefaced by "f" and all
// lowercase, by popular demand.

NtpSREventSummary evthdr; // sr summary data
NtpSRShieldSummary vetohdr; // veto shield summary data
NtpSRCosmicRay crhdr; // cr data, filled from last recons. trk ve
NtpSRDmxStatus dmxstatus; // status of demux
NtpSRDetStatus detstatus; // status of detector
NtpMCSummary mchdr; // mc summary data
NtpMCPhotonResult photon; // summary data from photon transport mc
NtpMCDetSimResult detsim; // summary data from det sim mc

// Standard reconstruction arrays
TClonesArray* vetostp; //-> array of shield strips of type NtpSRSh
TClonesArray* stp; //-> array of strips of type NtpSRStrip
TClonesArray* slc; //-> array of slices of type NtpSRSlice
TClonesArray* clu; //-> array of showers of type NtpSRCluster
TClonesArray* shw; //-> array of showers of type NtpSRShower
TClonesArray* trk; //-> array of tracks of type NtpSRTrack
TClonesArray* evt; //-> array of events of type NtpSREvent

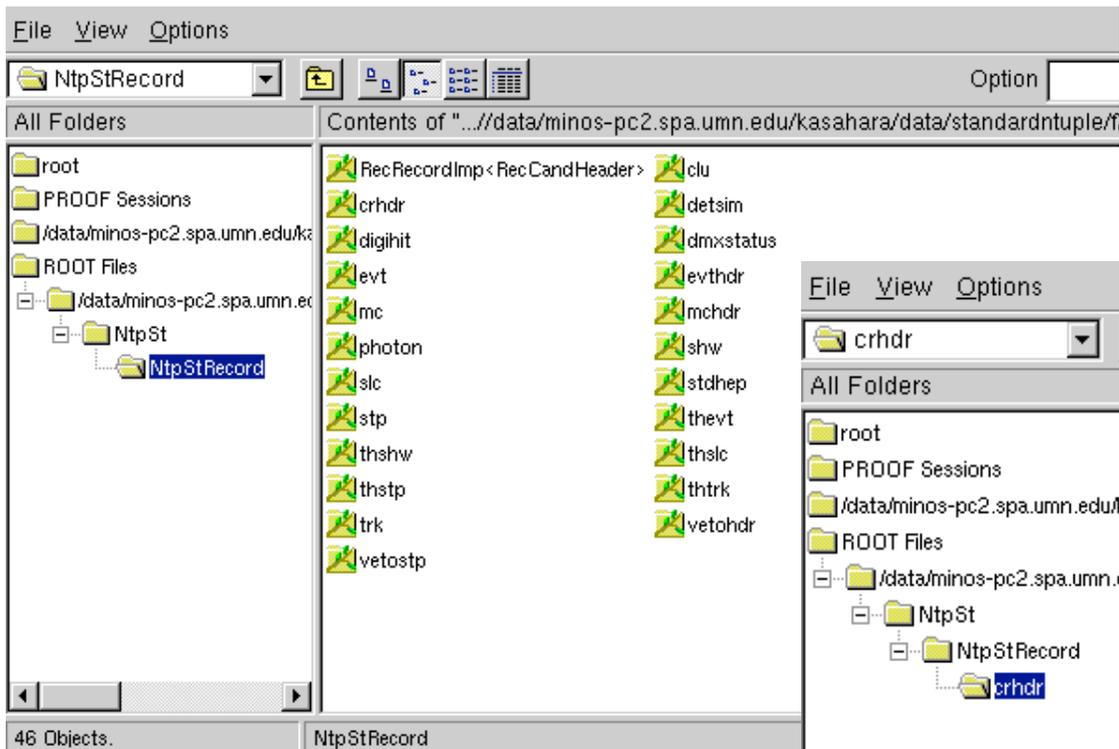
// MC truth arrays
TClonesArray* mc; //-> array of NtpMCTruth
TClonesArray* stdhep; //-> array of NtpMCStdHep
TClonesArray* digihit; //-> array of NtpMCDigiScintHit

// Truth helper arrays
TClonesArray* thstp; //-> array of NtpTHStrip
TClonesArray* thslc; //-> array of NtpTHSlice
TClonesArray* thshw; //-> array of NtpTHShower
TClonesArray* thtrk; //-> array of NtpTHTrack
TClonesArray* thevt; //-> array of NtpTHEvent

ClassDef(NtpStRecord,6)
};
```

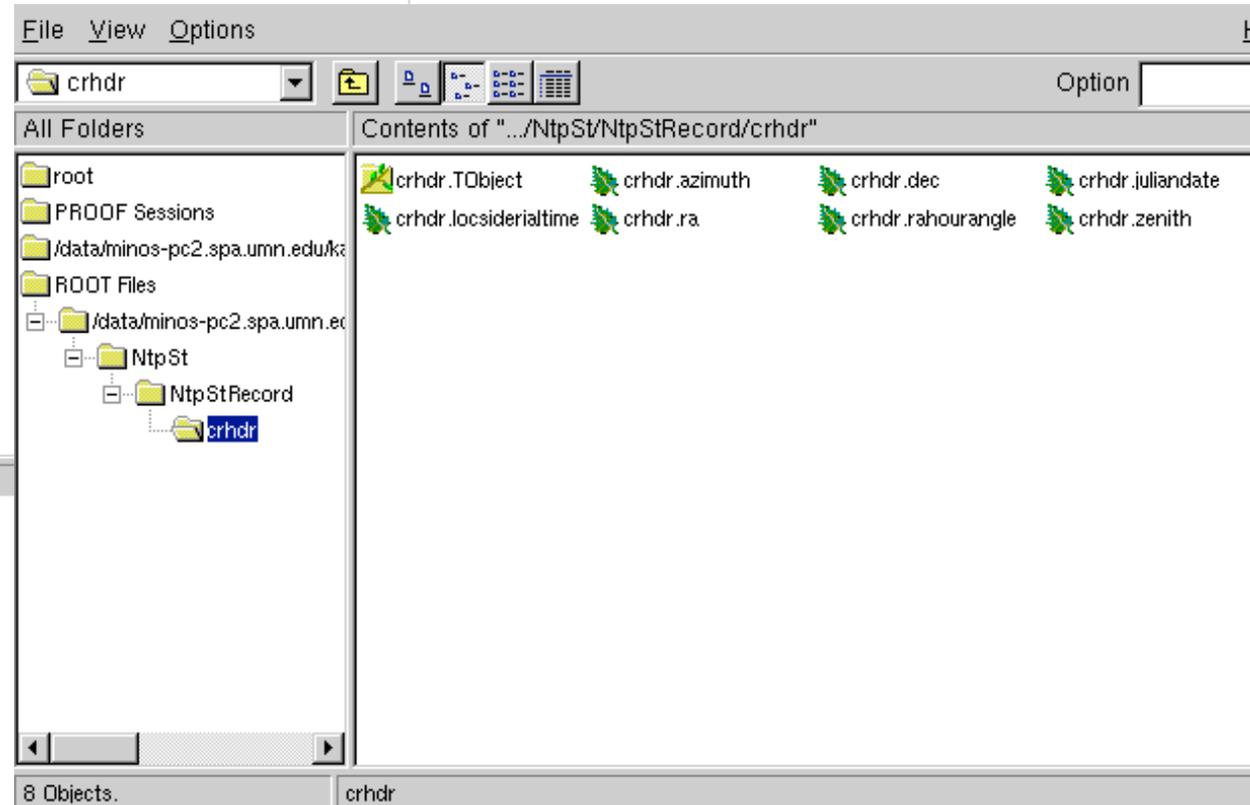
Standard Ntuples (NtpSt/SR/MC/TH), Structure and Content

Another way to view the same thing is through a ROOT TBrowser:



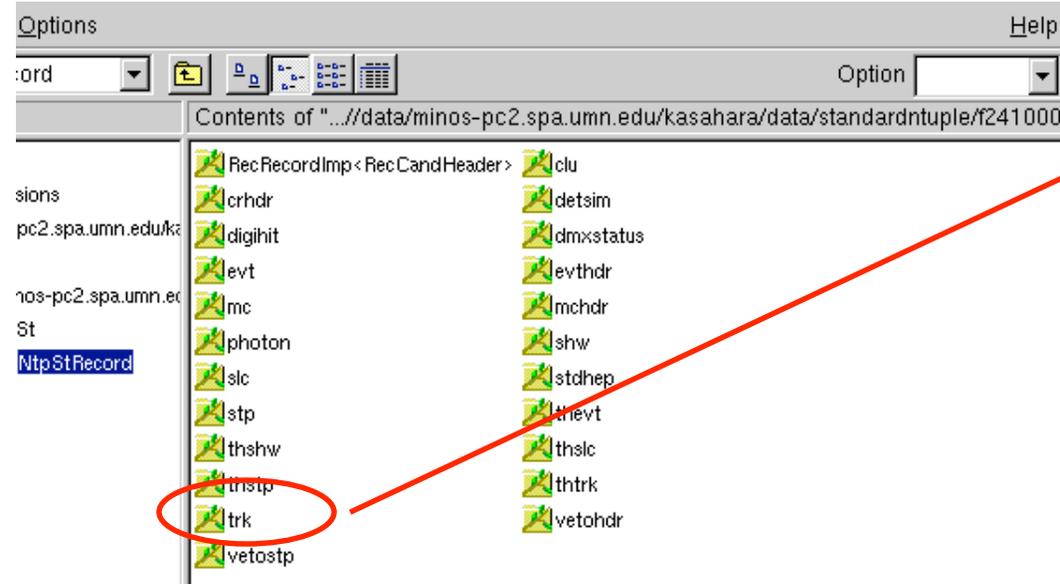
```
Float_t zenith; // zenith angle (degrees)
Float_t azimuth; // azimuthal angle measured easterly from north (degrees)
Float_t ra; // right ascension (degrees)
Float_t rahourangle; // right ascension (hours)
Float_t dec; // declination (degrees)
Double_t juliandate; // julian date (days since Jan 1, 4713 BC 12h UT)
Float_t locsiderialtime; // local sidereal time (hours)

ClassDef(NtpSRCosmicRay,1)
};
```



Standard Ntuples (NtpSt/SR/MC/TH), Structure and Content

Arrays of identical objects (TClonesArrays) are also stored on a branch split to basic data types:



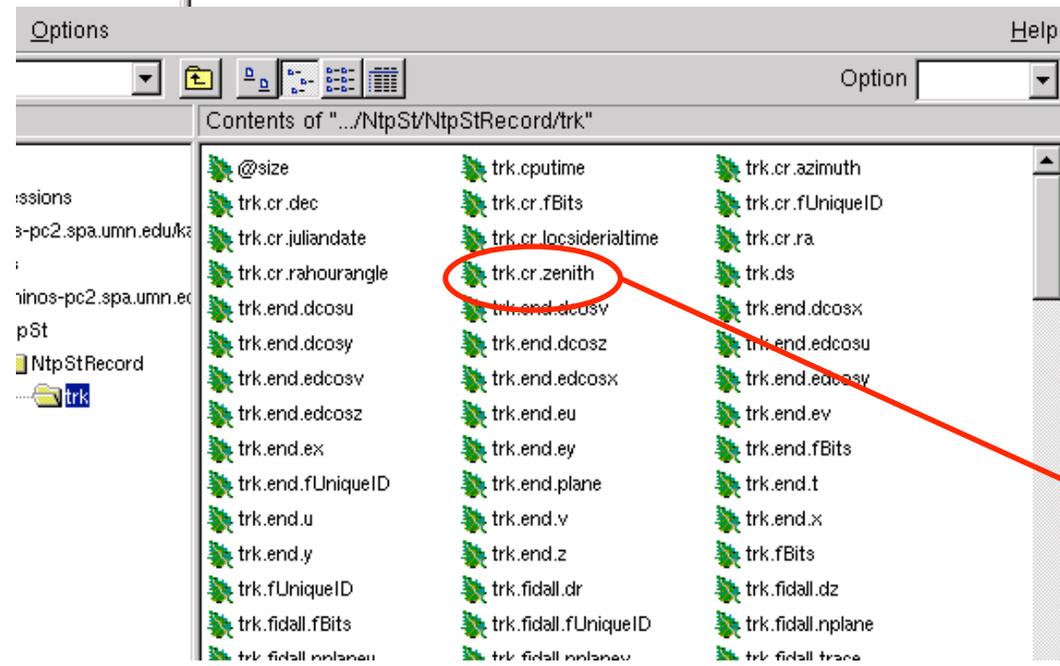
```

TClonesArray* stp; //-> array of strips of type NtpSRStrip
TClonesArray* slc; //-> array of slices of type NtpSRSlice
TClonesArray* clu; //-> array of showers of type NtpSRCluster
TClonesArray* shw; //-> array of showers of type NtpSRShower
TClonesArray* trk; //-> array of tracks of type NtpSRTrack
TClonesArray* evt; //-> array of events of type NtpSREvent

// MC truth arrays
TClonesArray* mc; //-> array of NtpMCTruth
TClonesArray* stdhep; //-> array of NtpMCStdHep
TClonesArray* digihit; //-> array of NtpMCDigiScintHit

// Truth helper arrays
TClonesArray* thstp; //-> array of NtpTHStrip
TClonesArray* thslc; //-> array of NtpTHSlice
TClonesArray* thshw; //-> array of NtpTHShower
TClonesArray* thtrk; //-> array of NtpTHTrack
TClonesArray* thevt; //-> array of NtpTHEvent

ClassDef(NtpStRecord,6)
};
    
```



([\\$SR_PUBLIC_CONTEXT/CandNtupleSR/NtpSRTrack.h](#)):

```

Double_t *stptcallt0; //[nstrip] T0 offset (sec) (west)
Float_t ds; // total path length of track from vertex to end (m)
Float_t range; // amount of material traversed from vertex to end (g/cm**2)
Float_t cputime; // CPU time to create track list which contains this track
NtpSRStripPulseHeight ph; // summed strip pulse height
NtpSRTrackPlane plane; // range of planes, timing used to determine start
NtpSRVertex vtx; // vertex coordinates
NtpSRVertex end; // coordinates at end plane
NtpSRVertex lin; // coordinates from linear fit
NtpSRFiducial fidvtx; // containment information for vertex
NtpSRFiducial fidend; // containment information for track end
NtpSRFiducial fidall; // containment information for distance of closest
// approach over all 3D points on the track
NtpSRTrackTime time; // track timing data
NtpSRMomentum momentum; // track momentum data
NtpSRFitTrack fit; // data from track fit
NtpSRCosmicRay cr; // cosmic ray data

ClassDef(NtpSRTrack,4)
};
    
```

T

Standard Ntuples (NtpSt/SR/MC/TH), How To Use

- Multiple approaches to accessing data from ntuples:
 - [ROOT TBrowser](#)
 - Useful for very simple displays of data.
 - Documentation of use in ROOT user manual.
 - [TTree::Draw, Scan, etc. executed at root prompt. \(example\)](#)
 - Useful for very simple displays of data.
 - Documentation of use in ROOT user manual and also Roy Lee's Numi note 860 which applies specifically to the MINOS ntuple style.
 - [TTree::MakeClass/MakeProxy](#)
 - MakeClass generates skeleton analysis class for tree.
 - Useful in cases where analysis of data is more complex.
 - Documentation of MakeClass use in TTree class documentation. MakeProxy is described on the Standard N-tuples web site with some examples of use.
 - [“C++” rootcint script. \(example\)](#)
 - Useful for complex analysis of ntuple data. Especially those which require imbedded iterative loops, e.g. looping over strips associated with tracks associated with events associated with the snarl.
 - A simple example is posted on the Standard N-tuple web site.
 - [Read ntuples back in through framework and analyze with a job module.](#)
 - Useful for complex analysis of ntuple data.
 - Ntuple records can be read in like any other record type back into the job framework. Specify input stream “NtpSt”.
 - Brian's AnalysisNtuple package takes this approach – the standard ntuple records are read back into the framework and a condensed analysis ntuple is generated as output.

Standard Ntuples (NtpSt/SR/MC/TH), How To Use

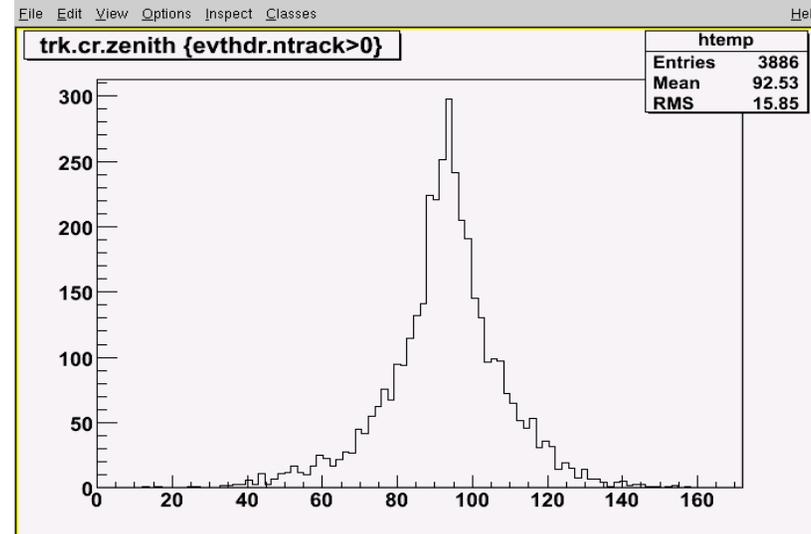
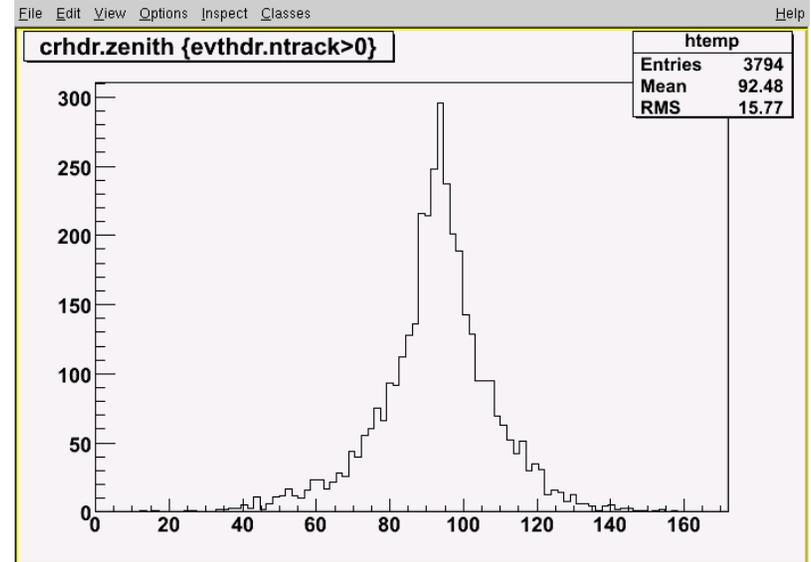
Using TTree methods at the ROOT prompt to analyze the ntuple TTree

- Basic ntuple-style approach to generating histograms & viewing data contents
- Commonly used TTree methods are:
 - `Draw(const char* varexp, const char* selection, Option_t* option, Long64_t nentries, Long64_t firstentry);`
 - Used to generate and draw histogram of data contained in TTree leaves selected using “varexp”. “selection” is used to apply a selection cut using TTree leaves.
 - `Scan(const char* varexp, const char* selection, Option_t* option, Long64_t nentries, Long64_t firstentry);`
 - Used to Print data, with arguments having same meaning as TTree::Draw for a leaf or leaves over multiple entries.
 - `Show(Long64_t entry, Int_t lenmax);`
 - Print all leaf values for the specified tree entry.
 - `Print(Option_t option);`
 - Print the tree branch structure, the number of entries, the number of bytes in each branch, etc.
- Documentation of use in the TTree class description and ROOT user manual

Standard Ntuples (NtpSt/SR/MC/TH), How To Use

Example using TTree commands:

```
root [0] f = new TFile("f24100022_0000_sntp.R1.16.root");
Warning in <TClass::TClass>: no dictionary for class VldContext is available
Warning in <TClass::TClass>: no dictionary for class VldTimeStamp is available
Warning in <TClass::TClass>: no dictionary for class RecRecord is available
Warning in <TClass::TClass>: no dictionary for class RecHeader is available
Warning in <TClass::TClass>: no dictionary for class RecPhysicsHeader is available
Warning in <TClass::TClass>: no dictionary for class RecDataHeader is available
Warning in <TClass::TClass>: no dictionary for class NtpStRecord is available
Warning in <TClass::TClass>: no dictionary for class RecRecordImp<RecCandHeader> is available
Warning in <TClass::TClass>: no dictionary for class RecCandHeader is available
Warning in <TClass::TClass>: no dictionary for class NtpSREventSummary is available
Warning in <TClass::TClass>: no dictionary for class NtpSRPulseHeight is available
Warning in <TClass::TClass>: no dictionary for class NtpSRPlane is available
Warning in <TClass::TClass>: no dictionary for class NtpSRDate is available
Warning in <TClass::TClass>: no dictionary for class NtpSRShieldSummary is available
Warning in <TClass::TClass>: no dictionary for class NtpSRCosmicRay is available
Warning in <TClass::TClass>: no dictionary for class NtpSRDmxStatus is available
Warning in <TClass::TClass>: no dictionary for class NtpMCSummary is available
Warning in <TClass::TClass>: no dictionary for class NtpMCPhotonResult is available
Warning in <TClass::TClass>: no dictionary for class NtpMCDetSimResult is available
Warning in <TClass::TClass>: no dictionary for class NtpSRShieldStrip is available
Warning in <TClass::TClass>: no dictionary for class NtpSRStrip is available
Warning in <TClass::TClass>: no dictionary for class NtpSRSlice is available
Warning in <TClass::TClass>: no dictionary for class NtpSRCluster is available
Warning in <TClass::TClass>: no dictionary for class NtpSRStripPulseHeight is available
Warning in <TClass::TClass>: no dictionary for class NtpSRShower is available
Warning in <TClass::TClass>: no dictionary for class NtpSRVertex is available
Warning in <TClass::TClass>: no dictionary for class NtpSRTrack is available
Warning in <TClass::TClass>: no dictionary for class NtpSRTrackPlane is available
Warning in <TClass::TClass>: no dictionary for class NtpSRFiducial is available
Warning in <TClass::TClass>: no dictionary for class NtpSRTrackTime is available
Warning in <TClass::TClass>: no dictionary for class NtpSRMomentum is available
Warning in <TClass::TClass>: no dictionary for class NtpSRFitTrack is available
Warning in <TClass::TClass>: no dictionary for class NtpSREvent is available
Warning in <TClass::TClass>: no dictionary for class NtpMCTruth is available
Warning in <TClass::TClass>: no dictionary for class NtpMCStdHep is available
Warning in <TClass::TClass>: no dictionary for class NtpMCDigiScintHit is available
Warning in <TClass::TClass>: no dictionary for class NtpTHStrip is available
Warning in <TClass::TClass>: no dictionary for class NtpTHSlice is available
Warning in <TClass::TClass>: no dictionary for class NtpTHShower is available
Warning in <TClass::TClass>: no dictionary for class NtpTHTrack is available
Warning in <TClass::TClass>: no dictionary for class NtpTHEvent is available
root [1] t = (TTree*)f->Get("NtpSt");
root [2] t -> Draw("crhdr.zenith","evthdr.ntrack>0");
<TCanvas::MakeDefCanvas>; created default TCanvas with name c1
root [3] t -> Draw("trk.cr.zenith","evthdr.ntrack>0");
root [4] █
```



Tutorial

S. Kasahara

Loop over track array entries is implicit. Can use [] notation to make explicit.

Standard Ntuples (NtpSt/SR/MC/TH), How To Use

Example using “C++” rootcint script to be run within loon

– Usage:

\$loon

loon[0].x testntp.C

```
[334]schubert@minos-pc2> more testntp.C
{
// Script to demonstrate read of ntuple file using a rootcint script
//
// Usage: .x thisscript.C
//
gSystem->Load("libCandNtupleSR.so");
gSystem->Load("libTruthHelperNtuple.so");
gSystem->Load("libMCNtuple.so");
gSystem->Load("libStandardNtuple.so");

// Create histo to fill, these will be available after script execution to
// draw
TH1D* pzenhist = new TH1D("primzenith","primary trk zenith",360,0.,360.);
TH1D* tzenhist = new TH1D("alltrkzenith","all trks zenith",360,0.,360.);

TFile* file = new TFile("f24100022_0000.sntp,R1,16.root","READ"); // open file

TTree* tree = (TTree*)(file->Get("NtpSt")); // extract tree
Int_t nentries = (Int_t)tree -> GetEntries();

// Prepare to loop over tree entries
// Tell the tree where to store retrieved data
NtpStRecord* strec = new NtpStRecord();
tree->SetBranchAddr("NtpStRecord",&strec);

// Loop over all tree entries
for ( Int_t ient = 0; ient < nentries; ient++ ) {
  tree -> GetEntry(ient); // data now accesible through strec
  Float_t pzen = strec->crhdr.zenith;
  // Because snarls without tracks will have negative zenith
  if ( pzen >= 0. ) pzenhist -> Fill(pzen);

  // Loop over all tracks
  Int_t ntrk = strec->evthdr.ntrack; // number of tracks in this event
  for ( int itrk = 0; itrk < ntrk; itrk++ ) {
    const NtpSRTrack* ntptrk = dynamic_cast<NtpSRTrack*>(strec->trk->At(itrk))
    Float_t tzen = ntptrk->cr.zenith;
    tzenhist -> Fill(tzen);
  }
  strec -> Clear();
}

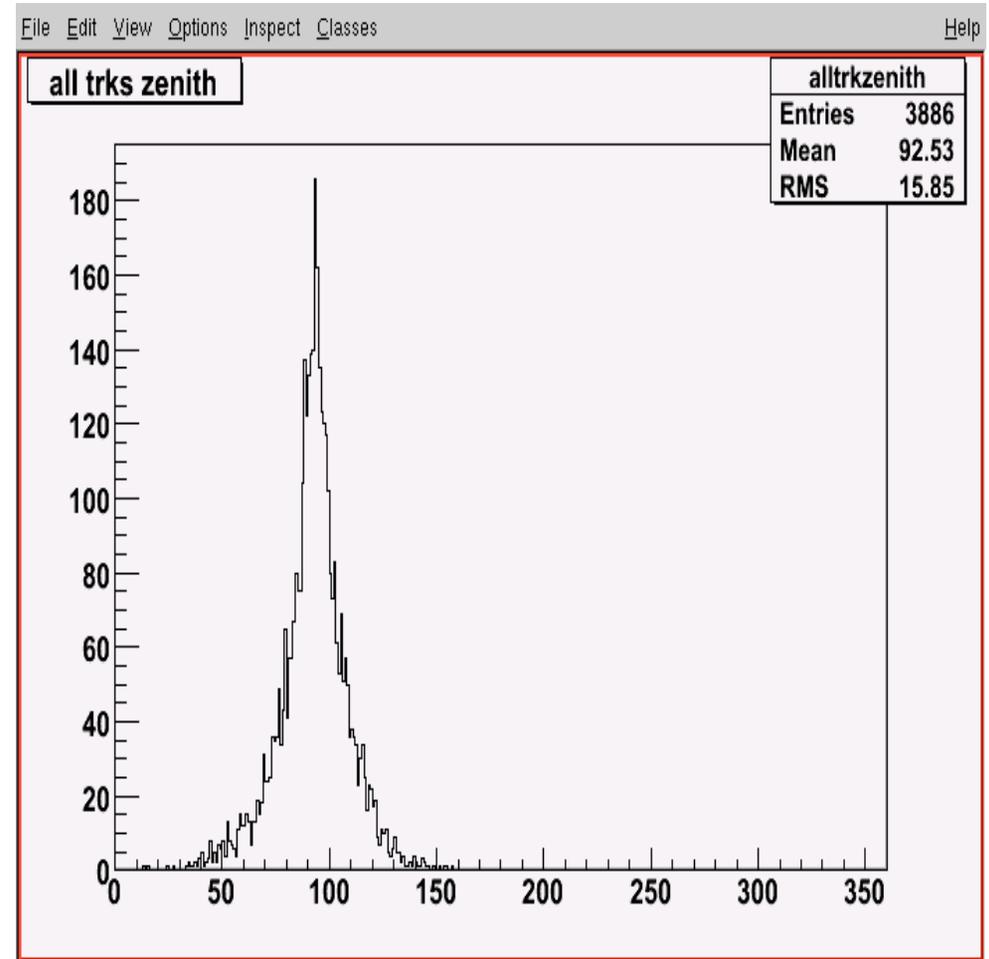
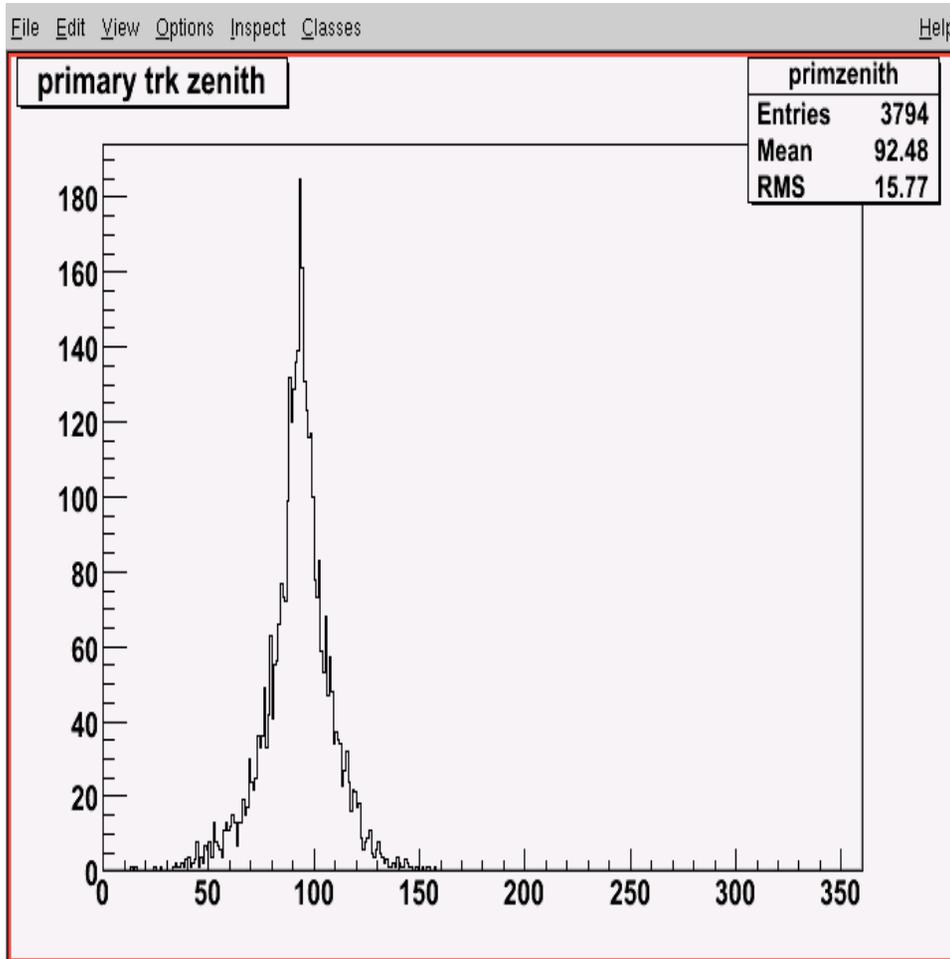
if ( strec ) delete strec;

pzenhist->Draw();
//tzenhist->Draw();
}
```

Standard Ntuples (NtpSt/SR/MC/TH), How To Use

- C++ rootcint script

```
loon [0] .x testntp.C  
<TCanvas::MakeDefCanvas>; created default TCanvas with name c1  
loon [1] tzenhist->Draw();  
loon [2] .q
```



Standard Ntuple (NtpSt/SR/MC/TH) Tutorial

Further References (Also listed on the Standard N-tuple web site):

- [The TTrees chapter of ROOT User's Guide and TTree class documentation.](#)
- [NUMI note 860 – Roy Lee's description of an older version of the current ntuple. The examples and description are still relevant.](#)
- [The Pete & Bernie show – introductory tutorials on video.](#)
- [The Ntuple example site off the Standard N-tuple web site.](#)
- [The ntuple chapter of the User Manual \(incomplete & probably out-of-date\)](#)
- [Nick's OO Companion Web Site](#)