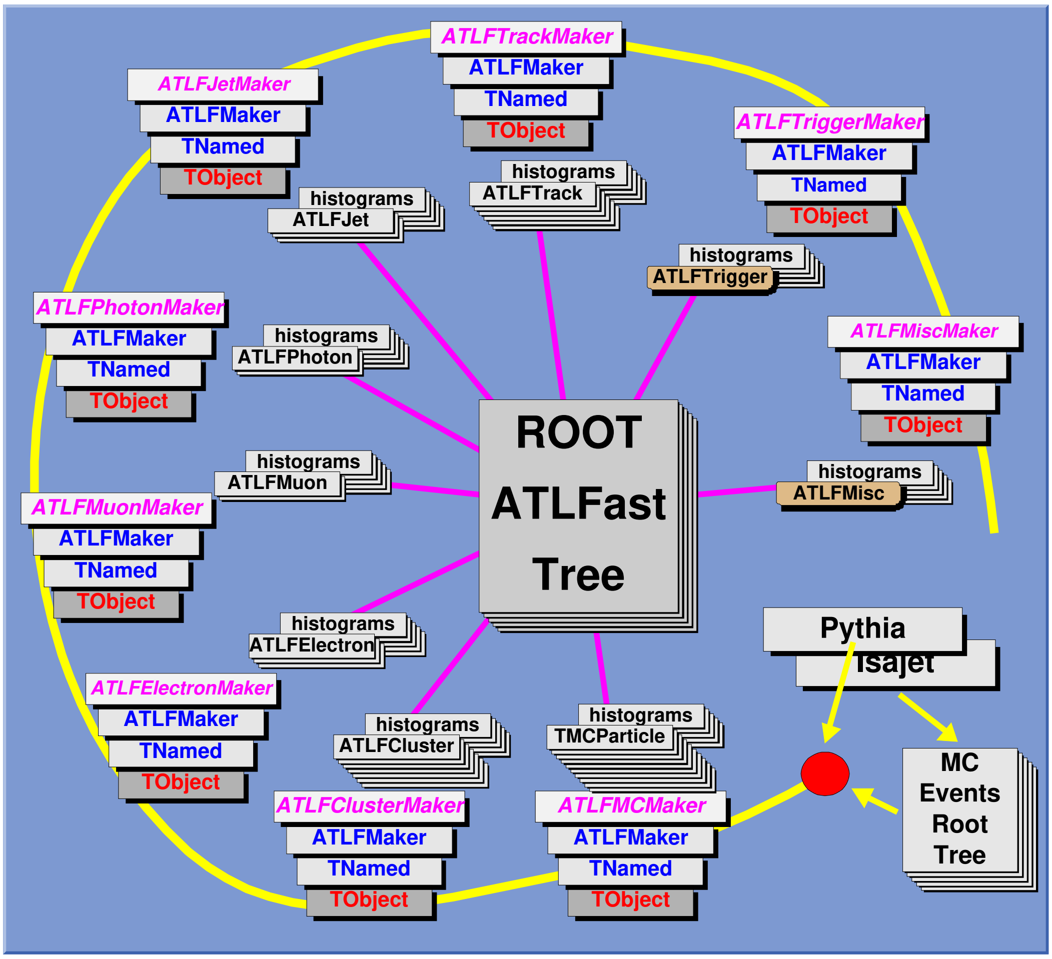
A fast simulation package  for ATLAS:  OO/C++ implementation

# Getting started with ATLFAST++

http:/root.cern.ch/root/html/atlfast/USER_Index.html




by R. Brun and E. Richter-Was

## ATLFAST - a fast simulation package for ATLAS

ATLFAST is a fast simulation package for ATLAS. Starting from the list of particles produced in generated events it provides the following output information after simulating the key aspects of the ATLAS detector response in a parametrised way:

- reconstructed jets (labelled as light jets, b-jets, c-jets, $\tau$-jets),
- isolated electrons and photons,
- isolated and non-isolated muons,
- missing transverse energy,
- a simplified trigger response,
- charged track parameters (for B-physics studies mostly)

ATLFAST-B is foreseen to be used on the events produced with ATLFAST:

- to randomly simulate b-tagging of jets labelled by ATLFAST.
- to provide a $p_T$ dependent recalibration of jet energies.
- to randomly simulate $\tau$-tagging and $\tau$-veto.

The OO/C++ version is derived from the original Fortran package and is based on Root. More information on the Fortran implementation can be obtained from Elzbieta Richter-Was (**erichter@hpjmiady.ifj.edu.pl**

The HTML documentation of the C++ implementation is available from

**http://root.cern.ch/root/html/atlfast/USER_Index.html**

The source files and test programs are available via anonymous ftp from **hpbrun.cern.ch** from

**public/atlfast/atlfast.tar.gz.**

Both implementations have the same functionality with one exception: The Event Display is available only for the C++ version.

An exhaustive document (about 70 pages) describing the principles of the algorithm, parameter sets, physics benchmark results, examples of analysis is distributed with the Fortran version (NOTE-1XY.ps). The present document is meant to explain the OO/C++ implementation, assuming that the reader is already familiar with the Fortran version of the program.


### Acknowledgements

Fons Rademakers has been an active participant in the design and evaluation phase . He is the author of the Makefiles for all Unix platforms.

Leanne Guy has implemented the track resolution algorithm in ATLFTrackMaker. She has actively participated to the testing phase of the C++ version.

E. Richter-Was would like to acknowledge the constant help from Piotr Golonka, a student of the Academy of Metallurgy of Cracow, who prepared the wrapper for TPythia class, helped to settle the ATLFast-B code, provided analysis code examples and who was always willing to explain mysteries of C++. Many warm thanks to Daniel Froidevaux for his constant support and advice, to Silvia Resconi for preparing an example with A, H $\rightarrow \tau\tau$ analysis and for being the "first user" of the package and to Fido Dittus for

## Conventions

ATLFast follows the coding guide lines defined by the ATLAS OO group.

These conventions can be quickly summarised:

- Data member names are of the form **m_MemberName**
- Member functions to set a data member are of the form **SetMemberName**
- Member functions to get the value of a member are **MemberName()**.
- All ATLFast classes start with the 4 letters **ATLF**.
- All Root classes start with **T**.
- Coding style follows the ATLAS recommendations.

## Setting environments

Some environment variables must be set before executing root and atlfast.

See **URL http://root.cern.ch/root/Availability.html**

On HP-UX, before executing the root module, you must set the library path:

**export SHLIB_PATH=$SHLIB_PATH:$ROOTSYS/lib** (in bash or ksh)

**setenv SHLIB_PATH ${SHLIB_PATH}:${ROOTSYS}/lib** (in csh or tcsh)

where $ROOTSYS is an environment variable pointing to the ROOT directory.

For example, if you use the HPUX-10 AFS version (see below), you should set:

**export ROOTSYS=/afs/cern.ch/na49/library/local/ROOT/hpux10/root** (inbash or ksh)

**setenv ROOTSYS /afs/cern.ch/na49/library/local/ROOT/hpux10/root** (incsh or tcsh)

Add the path **$ROOTSYS/bin** to your search path.

To run the program just type: **root.**

In the same way define **$ATLFAST** to point to the atlfast directory and **$PYTHIA** to point to the pythia directory. For more details, see the chapter "**Installation Procedure**" and the README file provided in the distribution kit.

## Availability

The ROOT framework is available on all Unix platforms and also on Windows NT and Windows 95.

ATLFAST is available on the following systems: HP-UX, Linux, SGI-IRIX, Sun Solaris, Alpha-Unix and Windows-NT.

More details on the local implementation can be found in the README files provided with the distribution kit.

## Class ATLFast

This class controls the execution flow and also the Input/Output. The idea is to use independent modules written by different users or teams, whilst communicating via a standardised object bus. This model is not specific to ATLFast and the picture shown in the cover page could be applied to nearly all production and analysis programs.

A program is in general a pipeline of modules. These modules can be added dynamically, or, the order of execution in the pipeline may be changed. In the case of ATLFast, these modules are called "Makers". The implemented Makers derive all from a common base class ATLFMaker. This class provides a common interface and defines some very simple and basic rules to be followed by all makers. The list of Makers is built by the ATLFast object constructor. We have only one instance of this object and we provide a global pointer called "gATLFast" that can be used for fast access to any parameter or object created in the program. The current implementation of ATLFast defines the following Makers:

```
m_MCMaker        = new ATLFMCMaker("MCMaker","Make MC events");
m_ClusterMaker   = new ATLFClusterMaker("ClusterMaker","Make ATLFast clusters");
m_MuonMaker      = new ATLFMuonMaker("MuonMaker","Make ATLFast muons");
m_ElectronMaker  = new ATLFElectronMaker("ElectronMaker","Make ATLFast electrons");
m_PhotonMaker    = new ATLFPhotonMaker("PhotonMaker","Make ATLFast photons");
m_JetMaker       = new ATLFJetMaker("JetMaker","Make ATLFast jets");
m_MiscMaker      = new ATLFMiscMaker("MiscMaker","Make ATLFast miscelaneous");
m_TriggerMaker   = new ATLFTriggerMaker("TriggerMaker","Make ATLFast trigger");
m_TrackMaker     = new ATLFTrackMaker("TrackMaker","Make ATLFast tracks");
```

These Makers objects are inserted into the linked list of Makers created in the ATLFast object (**member m_Makers**). This list of makers can be accessed via **gATLFast->Makers()**.

In all the phases of the program, Initialisation, Event loop or Termination, the corresponding Makers functions are always executed in the order defined in the list. For example, the code to process one event looks like

```
void ATLFast::Make(Int_t event)
{
//*  Loop on all makers
     TIter next(ListMakers());
     ATLFMaker *maker;
     while (maker = (ATLFMaker*)next()) {
        maker->Make(event);
     }
}
```

Each Maker is responsible for creating its own branch of the Tree. The following table shows the list of makers currently implemented. The default option to Save the Maker info in the Tree is given.

| Maker name | Save in Tree |
|---|---|
| MCMaker | Yes (partial only) |
| ClusterMaker | No |
| ElectronMaker | Yes |
| MuonMaker | Yes |
| PhotonMaker | Yes |
| JetMaker | Yes |
| TrackMaker | Yes |
| TriggerMaker | Yes |
| MiscMaker | Yes |

The ROOT system uses **dynamic linking**. This has two advantages:

- link time is considerably reduced (non existant)
- the link executable modules are smaller

We provide a macro **Loadlibs** to dynamically link shared libraries the **libAtlfast.so, libAtlfastg.so** and **ROOT/PYTHIA libraries**.

**macro loadlibs**

```
{
 gSystem->Load("$PYTHIA/libPythia.sl");
 gSystem->Load("libEG.sl");
 gSystem->Load("libEGPythia.sl");
 gSystem->Load("libAtlfast.sl");
 gSystem->Load("libAtlfastg.sl");
}
```

### Executing ATLFast

An example of execution is given in the macro "**umain.C**" below. You can execute this macro interactively or in batch. To execute the macro interactively, you run the standard executable of Root

```
root
Root > .x umain.C     //this will execute the code in macro umain.C
Root > .q              //quit from Root
```

To execute the macro in batch mode;

```
root -b -q umain.C >program.log
```

The above command invokes root in batch mode, executing the macro **umain.C** and saving the standard output in the file **program.log**.

**macro umain.C**

```
void umain(Int_t nevents=10000)
{
    //* Dynamically link ATLAS shared libraries
  gROOT->Macro("loadlibs");

   //* Open the Root output file
  TFile file("atlas.root","recreate","ATLFast root file");

   //* Create control object to run ATLFast
   //* Create Makers histograms and the Root tree
  ATLFast atlfast("atlfast");
  atlfast.Init();
  atlfast.MakeTree();

   //* Read a macro with interpreted User class
  gROOT->LoadMacro("user.C");
  User user;

   //* Loop on events
  for (Int_t i=0; i<nevents; i++) {
    atlfast.Make(i);      // Generate and Reconstruct event
    atlfast.FillTree();   // Fill Root tree
    user.FillHistograms();
    atlfast.Clear();      // Clear reconstructed event lists
  }

  //* Save atlfast, Makers, histograms and Tree header
  atlfast.Write();
}
```

In the makefile distributed with the system, one has the possibility to create a compiled main program instead of executing **umain.C** (see **gmake atlroot** or **gmake atlfast**). The macro **umain.C** is interpreted at run time. This macro calls another macro (**user.C**). The macro **user.C** can be customised by each user to set different run conditions, add new histograms, etc.

**macro user.C**

```
{
  // change ATLfast default parameters

   gATLFast->SetSmearing(1);
   gATLFast->ClusterMaker()->SetGranBarrelEta(0.1);
   gATLFast->ClusterMaker()->Save(0);
   gATLFast->PhotonMaker()->SetMinPT(5.);
   gATLFast->TriggerMaker()->SetMuoEtaCoverage(2.5);
   gATLFast->MCMaker()->Save(1);      //give 2 to save all particles
   gATLFast->TrackMaker()->Save(0);   // give 1 to save tracks
}
```

I

## The ROOT output Tree

The following information is saved in the Root data base.

- Program parameters. The **object ATLFast** as well as all **Maker objects** are saved. An analysis program knows the exact parameters used to produce the current data base.
- The list of histograms from each maker.
- The list of objects produced for each event. This event dependent information is stored in the form of a Root Tree (see **http://root.cern.ch/root/html/TTree.html** )
- A **Tree** is an extension of the PAW ntuples. The **Tree** (**called "T" in Atlfast**) is automatically built from the class model. The generation of branches is automatic for each Maker. In turn, each attribute of the result objects goes into a separate branch. This organisation has the following advantages:
  - It preserves the original data model.
  - The organisation of containers in the data base is automatic and optimal for data analysis.
  - The data compression is optimum because branch buffers contain homogeneous data types.

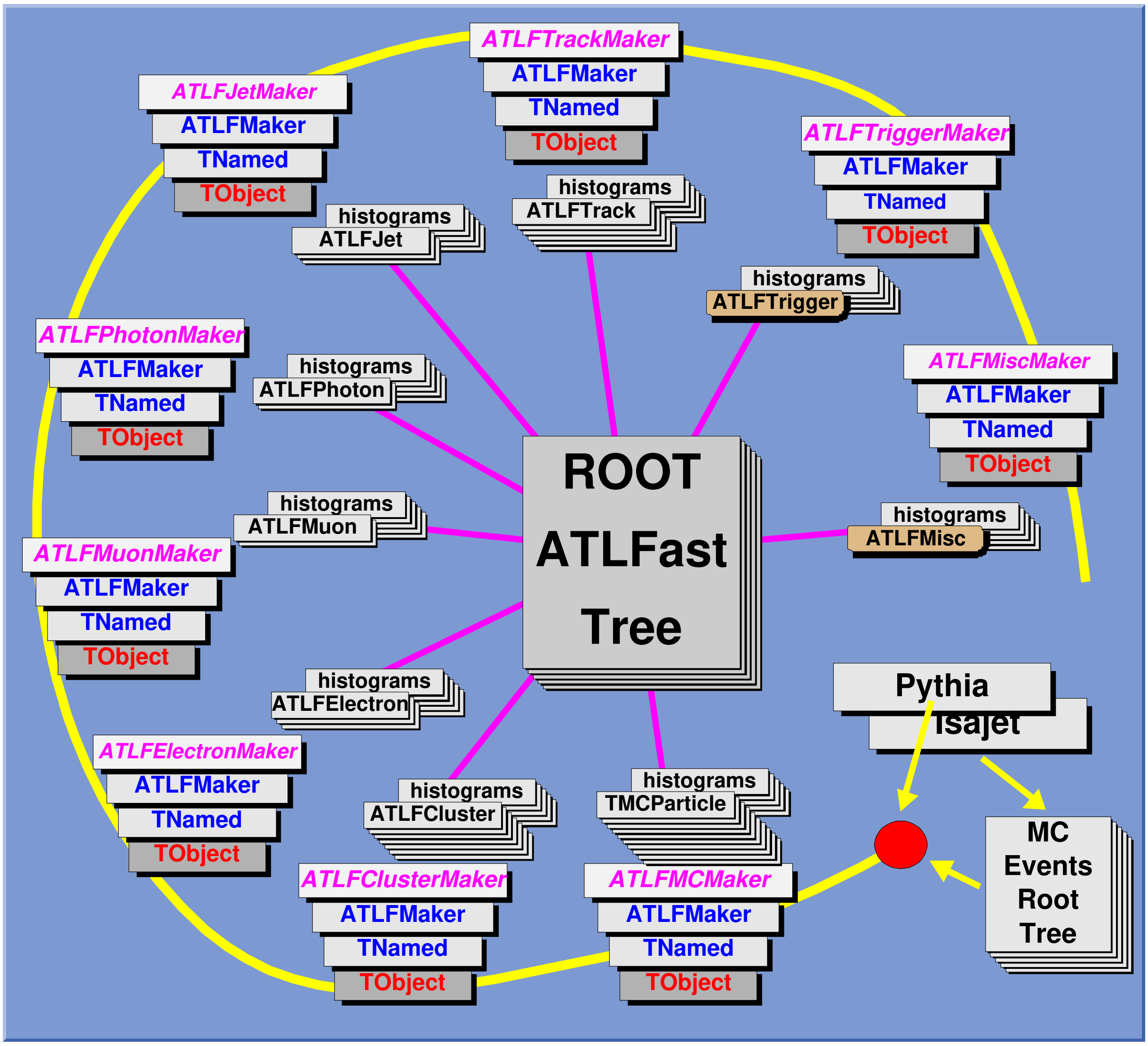
The Root data base is portable across heterogeneous architectures (32, 64 bits, big or little endian byte ordering). You can use binary FTP, NFS/AFS or access remote files (using the **rootd file server**).

## ATLFMaker: General principles

This is the base class for all **Makers**. it includes the following functions;

- The constructor automatically adds the maker to the list of **Makers** in the **ATLFast object**
- The **Browse** function provides a default functionality (usually sufficient) to browse the products of this **Maker**.
- The **Clear** function called at the end of each event removes the event objects from the **TClonesArray** list.
- The **Draw** function controls the **Event display** for this **Maker**.
- The MakeBranch function is called by **ATLFast::MakeTree**. It creates the **Tree branch** for the **TClonesArray** or **TObject** result product of this **Maker.**
- The **SetChainAddress** function is called by **ATLFast::SetChainAddress** in case a list of Root files must be analyzed. It takes care of setting the appropriate **TClonesArray** address when reading each file in the chain.

All above functions do not need to be redefined in the derived classes.



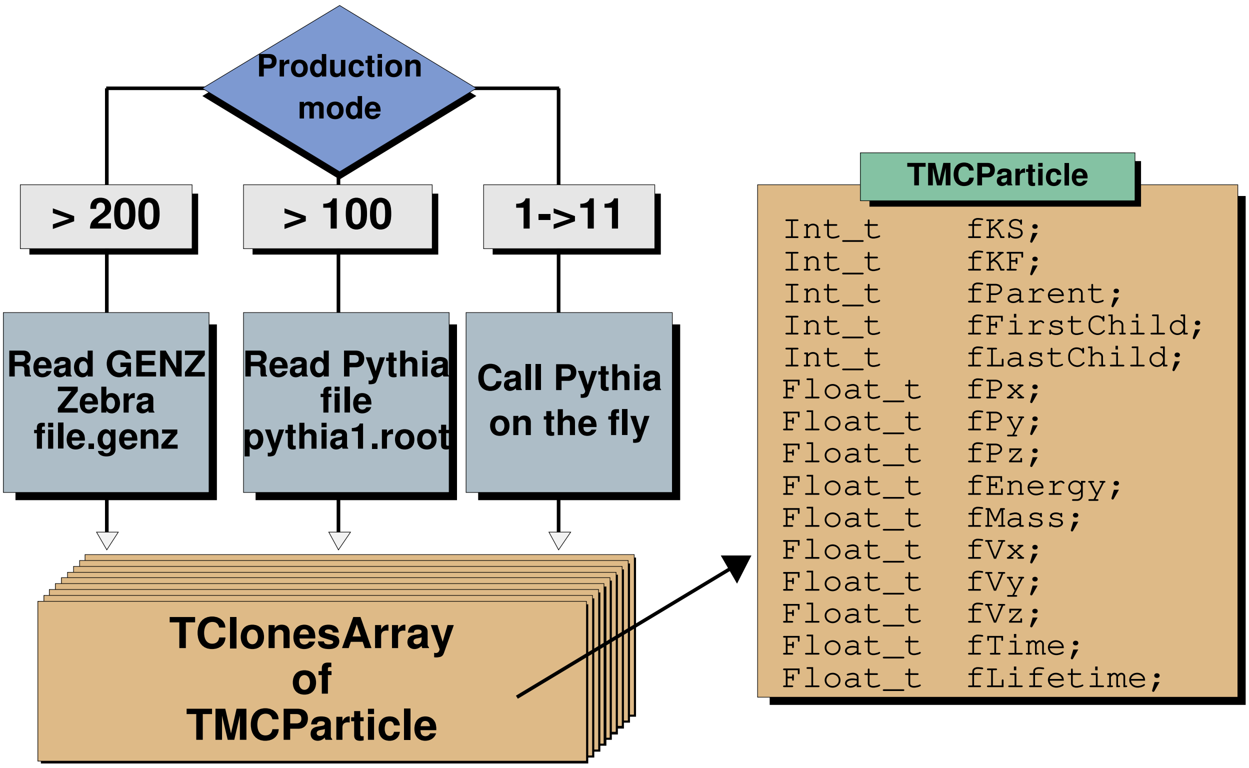
## ATLFMCMaker

This class organises the generation of the physics process via the event generator Pythia. The interface to the Pythia package (Fortran) uses the services of the **Root class TPythia**, see:

**`http://root.cern.ch/root/html/TPythia.html`**

The results of Pythia are stored in a common block /LUJETS/. The function **TPythia::GetPrimaries** imports this common block into a list of particle objects (**Root class TMCParticle**).

The Pythia production mode and hadronisation mode can be controlled via **ATLFMCMaker::SetProductionMode** and **SetHadronisationMode**. This class also supports the reading of Pythia events already stored in a Root file or in a Zebra/GENZ file.




**Example of loop on all generated particles**

```
ATLFMCMaker *mcmaker = gATLFast->MCMaker();
TClonesArray *particles = mcmaker->Fruits();
TH1F *hE = new TH1F("hE","Tracks_energy",100,0,100);
TMCParticle *particle;
Int_t nparticles = particles->GetEntriesFast();
for (Int_t i=0;i<nparticles;i++) {
   particle = (TMCParticle*)particles->At(i);
   hE->Fill(particle->GetEnergy());
}
```

| ATLFMCMaker data members | Default value | Title |
|---|---|---|
| Int_t m_ProductionMode | 1 | Production mode (see Init) |
| Int_t m_HadronisationMode | 3 | Hadronisation mode (see Init |
| Int_t m_RndNumber | 51111 | RndNumber(see Init) |
| Int_t m_Nparticles | | Number of MC particles |
| Int_t m_Nstop | | Last parton number |
| Int_t m_Nstart | | First non-parton number |
| Int_t m_Nstables | | Number of stable particles |
| TObject *m_Generator | | Pointer to the event generator<br>Pointer to TPythia object. (default) |

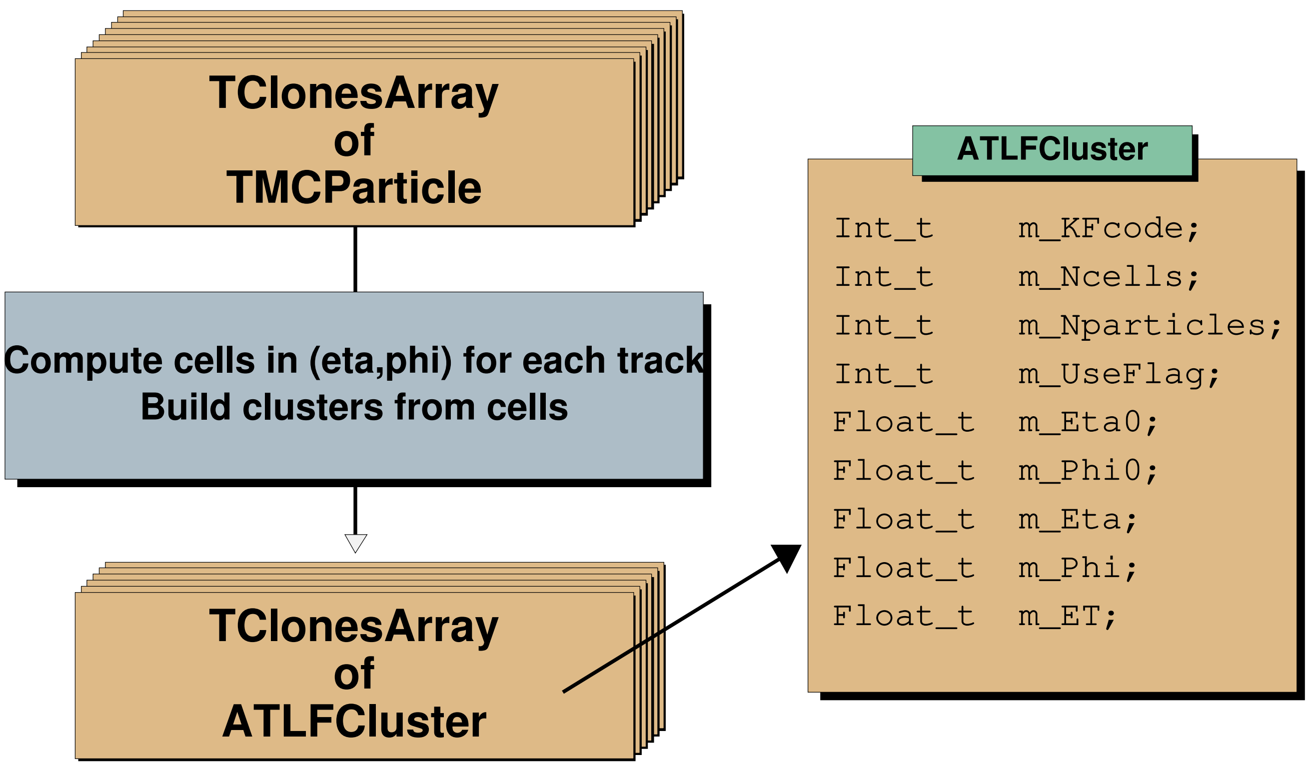
| Predefined production modes | Process |
|---|---|
| m_ProductionMode = 1 | $H \rightarrow \gamma\gamma$ (default) |
| m_ProductionMode = 2 | $Z \rightarrow e^+e^-$ |
| m_ProductionMode = 3 | $Z \rightarrow \mu^+\mu^-$ |
| m_ProductionMode = 4 | $H \rightarrow e^+e^-e^+e^-$ |
| m_ProductionMode = 5 | $H \rightarrow e^+e^-\mu^+\mu^-$ |
| m_ProductionMode = 6 | $H \rightarrow \mu^+\mu^-\mu^+\mu^-$ |
| m_ProductionMode = 7 | WH , $H \rightarrow b\bar{b}$ |
| m_ProductionMode = 8 | WH , $H \rightarrow gg$ |
| m_ProductionMode = 9 | WH , $H \rightarrow u\bar{u}$ |
| m_ProductionMode = 10 | A, $H \rightarrow \tau\tau$ |

| Predefined hadronization modes | Description |
|---|---|
| m_HadronisationMode = 0 | Only hard process (HP) |
| m_HadronisationMode = 1 | HP + Initial State Radiation (ISR) |
| m_HadronisationMode = 2 | HP+ISR+Final Stae Radiation (FSR) |
| m_HadronisationMode = 3 | HP+ISR+FSR+Hadronisation (HD) (default) |
| m_HadronisationMode = 4 | HP+ISR+HD |
| m_HadronisationMode = 5 | HPIFSR+HD |

## ATLFClusterMaker

Deposites the energy of generated particles into the calorimeter $\Delta\eta$ x $\Delta\phi$ cells ( magnetic field effects included) and reconstructs clusters for a given cone and minimum $E_T$. The granularity in $\eta$, $\phi$ , and the reconstruction cone $\Delta R$ can be changed via the functions **SetGranBarrelEta**, **SetGranBarrelPhi**, **SetDeltaRBarrel1**, etc.

All parameters (see Table on the next page) can be changed via the corresponding **Set** functions. The reconstructed clusters are stored as individual objects (**class ATLFCluster**) in a **TClonesArray** list of clusters.




**Example of loop on all reconstructed clusters**

```
ATLFClusterMaker *clustermaker = gATLFast->ClusterMaker();
TClonesArray *clusters = clustermaker->Fruits();
TH1F *hEta = new TH1F("hEta","Rapidity",100,-5,5);
ATLFCluster *cluster;
Int_t nclusters = clusters->GetEntriesFast();
for (Int_t i=0;i<nclusters;i++) {
   cluster = (ATLFCluster*)clusters->At(i);
   hEta->Fill(cluster->Eta());
}
```

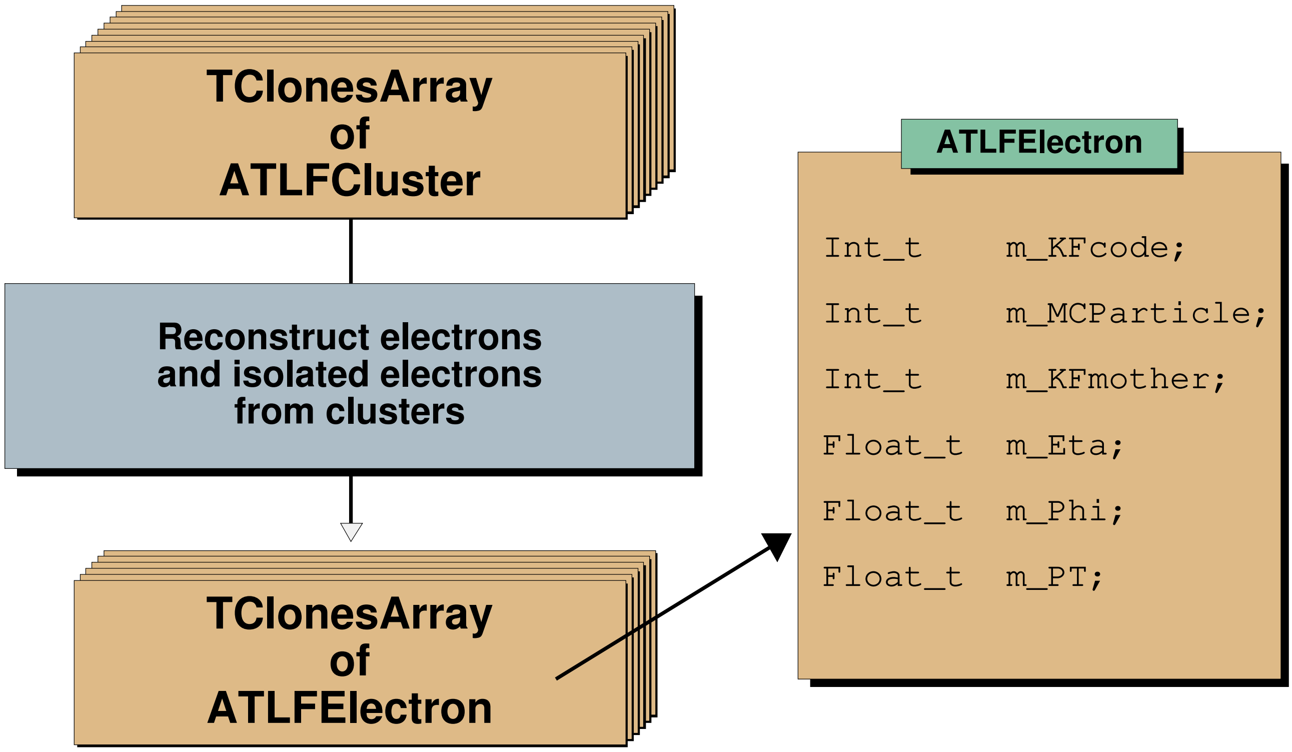
| ATLFClusterMaker data member | Default value | Title |
|---|---|---|
| Int_t m_Ncells; | | Number of cells |
| Int_t m_Nclusters; | | Number of clusters |
| Float_t m_MinETCalo; | 10.0 | Minimum E_T for calo cluster |
| Float_t m_DeltaRBarrel1; | 0.4 | Cone Delta R for barrel set 1 |
| Float_t m_DeltaRForward1 | 0.4 | Cone Delta R for forward set 1 |
| Float_t m_DeltaRBarrel2; | 0.7 | Cone Delta R for barrel set 2 |
| Float_t m_DeltaRForward2 | 0.7 | Cone Delta R for forward set 2 |
| Float_t m_EtaCoverage | 5.0 | Rapidity coverage for calorimeter |
| Float_t m_MinETInitiator; | 1.5 | Min E_T for cluster initiator |
| Float_t m_MinPTBfield; | 0.5 | Min p_T for B-field |
| Float_t m_MinETCell; | 0.0 | Min E_T for cell |
| Float_t m_BarrelForwardEta; | 3.0 | Barrel/fwd transition in eta in calo |
| Float_t m_GranBarrelEta; | 0.1 | Granularity in eta(barrel),2X in fwd |
| Float_t m_GranBarrelPhi; | 0.1 | Granularity in phi(barrel),2X in fwd |
| Bool_t m_CellsEsharing; | 0 | Cells energy sharing in clusters on/off |

| ATLFClusterMaker standard histograms | Title |
|---|---|
| TH1F *m_Multiplicity | clusters multiplicity |
| TH1F *m_DeltaPhi | B-field delta_phi vers p_T |
| TH1F *m_Counters | B-field counters vers p_T |
| TH1F *m_PhiClu; | delta_phi cluster-barry_centre_particle |
| TH1F *m_EtaClu | delta_eta cluster-barry_centre_particle |
| TH1F *m_DeltaR | delta_r cluster-barry_centre_particle |

### ATLFElectronMaker

Starting form the list of reconstructed clusterss, ATLFElectronMaker classifies electrons as either isolated or nonisolated. Electron energy is then smeared with parametrised resolutions. Calorimeter energy clusters associated with isolated electrons are discarded.

The reconstructed electrons are stored as objects (**class ATLFElectron**) into a **TClonesArray** list of electrons.




**Example of loop on all reconstructed electrons**

```
ATLFElectronMaker *electronmaker = gATLFast->ElectronMaker();
TClonesArray *electrons = electronmaker->Fruits();
TH1F *hPT = new TH1F("hPT","Transverse_Momentum",100,0,50);
ATLFElectron *electron;
Int_t nelectrons = electrons->GetEntriesFast();
for (Int_t i=0;i<nelectrons;i++) {
   electron = (ATLFElectron*)electrons->At(i);
   hPT->Fill(electron->PT());
}
```

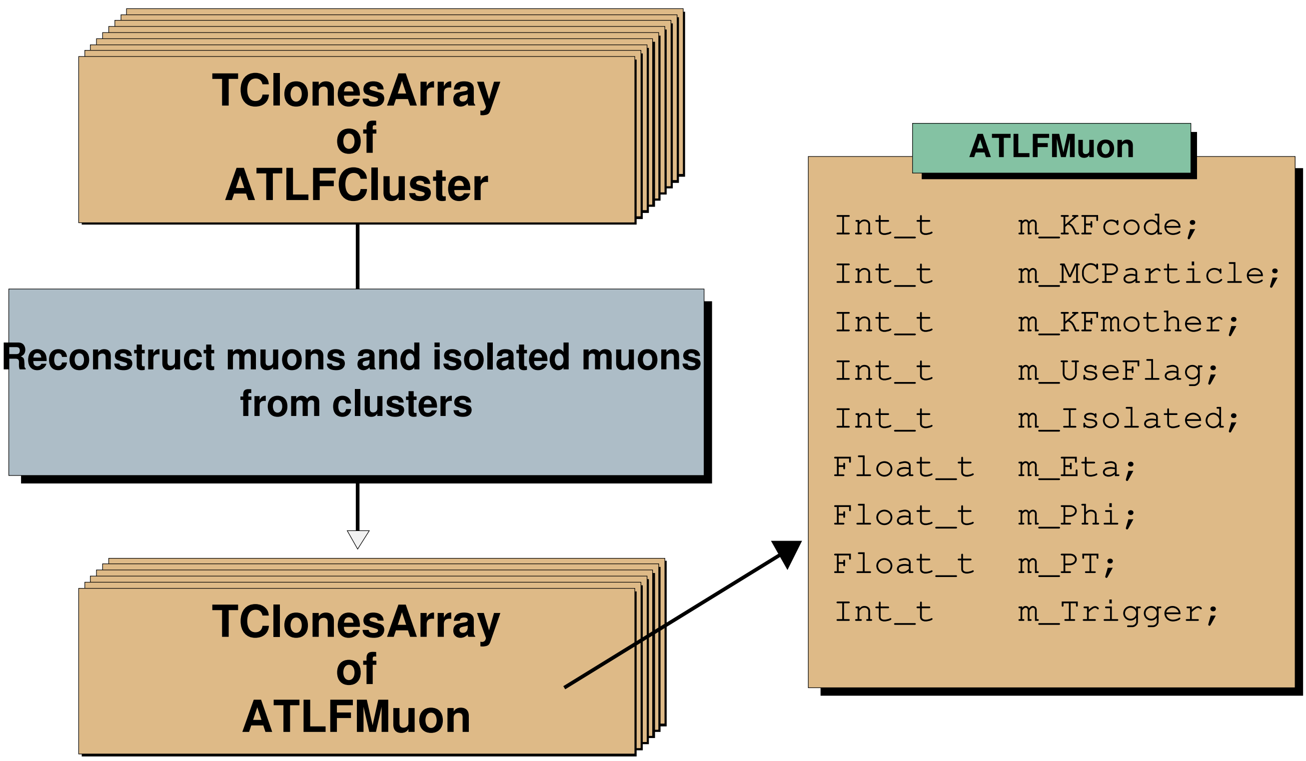
| ATLFElectronMake data members | Default value | Title |
|---|---|---|
| Int_t m_Nelectrons | | Number of electrons |
| Float_t m_MinPT | 5.0 | Minimum electron transverse momentun |
| Float_t m_MaxEta | 2.5 | Maximum electron eta |
| Float_t m_RconeMatch | 0.15 | Delta R cone for electron matching to cluster |
| Float_t m_RconeSep | 0.4 | Min cone R separation from clusters for electron isolation |
| Float_t m_RconeDep | 0.2 | R cone for energy deposition |
| Float_t m_MaxEdep | 10.0 | Max energy deposition for isolation |

| ATLFElectronMake standard histograms | Title |
|---|---|
| TH1F *m_Mult; | electron multiplicity |
| TH1F *m_MultHard; | electron multiplicity hard |
| TH1F *m_MultHardIsol; | electron multiplicity hard + isol |
| TH1F *m_PT; | (pT- pTcru) vers E electron ISOLATED |
| TH1F *m_Eta; | (eta-etacru) vers E electron ISOLATED |
| TH1F *m_Phi; | (phi-phicru) vers E electron ISOLATED |
| TH1F *m_Counter; | counter vers E electron ISOLATED |
| TH1F *m_Mass2e; | ee mass |
| TH1F *m_Mass2esubst; | ee mass subtract width |
| TH1F *m_Mass4e; | eeee mass |

### ATLFMuonMaker

Starting from the list of clusters, ATLFMuonMaker classifies muons as either isolated or nonisolated. Their $1/p_T$ value is then smeared with parametrised resolutions.

The reconstructed isolated and nonisolated muons are stored as objects (**class ATLFMuon**) into a **TClonesArray** list of muons.




**Example of loop on all reconstructed muons**

```
ATLFMuonMaker *muonmaker = gATLFast->MuonMaker();
TClonesArray *muons = muonmaker->Fruits();
TH1F *hPT = new TH1F("hPT","Transverse_Momentum",100,0,50);
ATLFMuon *muon;
Int_t nmuons = muons->GetEntriesFast();
for (Int_t i=0;i<nmuons;i++) {
   muon = (ATLFMuon*)muons->At(i);
   hPT->Fill(muon->PT());
}
```

| ATLFMuonMaker data members | Default value | Title |
|---|---|---|
| Int_t m_Nmuons | | Number of muons |
| Float_t m_MinPT | 6.0 | Minimum muon transverse momentum |
| Float_t m_MaxEta | 2.5 | Maximum muon eta |
| Float_t m_RconeSep | 0.4 | Min cone R separation from clusters for muon isolation |
| Float_t m_RconeDep | 0.2 | R cone for energy deposition |
| Float_t m_MaxEdep | 10.0 | Max energy deposition for iso-lation |

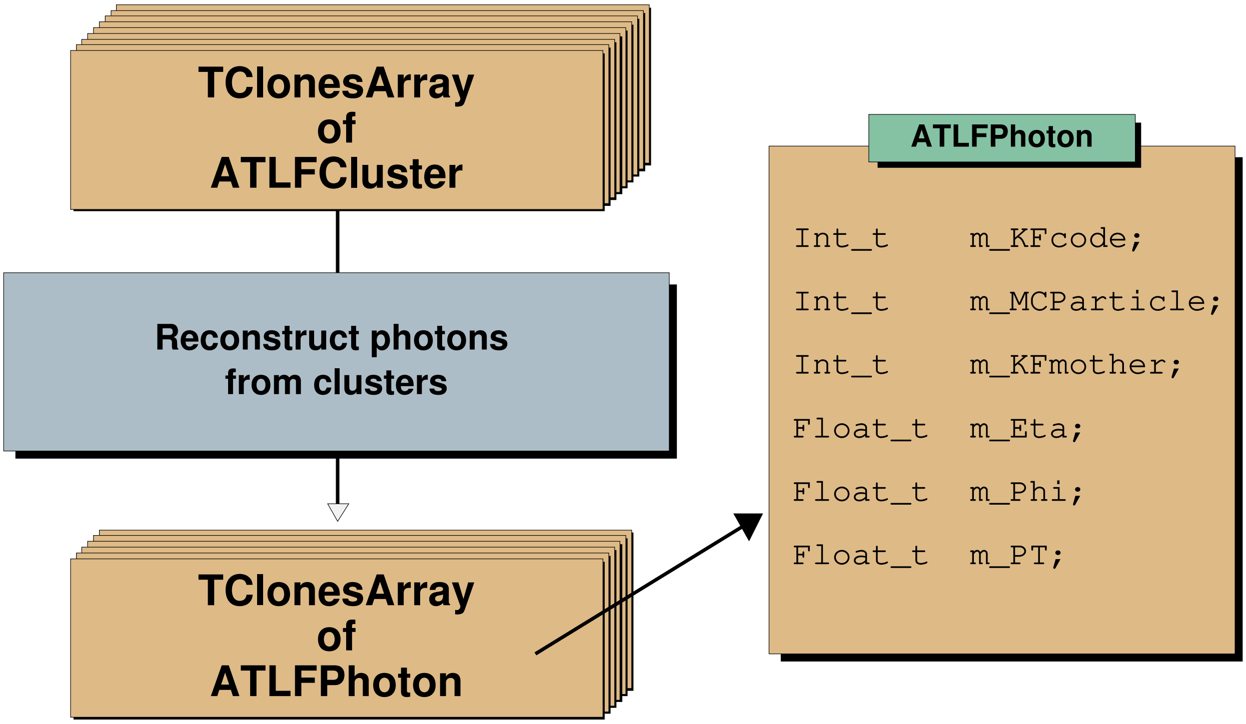
| ATLFMuonsMaker standard histograms | Title |
|---|---|
| TH1F *m_Mult | muon multiplicity NOISOLATED |
| TH1F *m_MultIsol | muon multiplicity ISOLATED |
| TH1F *m_MultHard | muon multiplicity hard |
| TH1F *m_MultHardIsol | muon multiplicity hard + isol |
| TH1F *m_PT | (pT- pTcru) vers pT muon ISOLATED |
| TH1F *m_Eta | (eta-etacru) vers pT muon ISOLATED |
| TH1F *m_Phi | (phi-phicru) vers pT muon ISOLATED |
| TH1F *m_Counter | counter vers pT muon ISOLATED |
| TH1F *m_Mass2mu | mumu mass |
| TH1F *m_Mass2musubst | mumu mass subtract width |
| TH1F *m_Mass4mu | 4mu mass |

### ATLFPhotonMaker

Starting form the list of reconstructed clusters, ATLFPhotonMaker classifiess photons as either isolated

or nonisolated, then smears their energy and angular position with parametrised resolutions. Calorimeter energy clusters associated with isolated photons are discarded.

The reconstructed isolated photons are stored as objects (**class ATLFPhoton** ) into a **TClonesArray** list of photons.




**Example of loop on all reconstructed photons**

```
ATLFPhotonMaker *photonmaker = gATLFast->PhotonMaker();
TClonesArray *photons = photonmaker->Fruits();
TH1F *hPT = new TH1F("hPT","Transverse_Momentum",100,0,50);
ATLFPhoton *photon;
Int_t nphotons = photons->GetEntriesFast();
for (Int_t i=0;i<nphotons;i++) {
   photon = (ATLFPhoton*)photons->At(i);
   hPT->Fill(photon->PT());
}
```

Protected data members of the class ATLFPhotonMaker.

| ATLFPhotonMaker data members | Default value | Title |
|---|---|---|
| Int_t m_Nphoton | | Number of photons |
| Float_t m_MinPT | 5.0 | Minimum photon transverse momentunm |
| Float_t m_MaxEta | 2.5 | Maximum photon eta |
| Float_t m_RconeMatch | 0.15 | Delta R cone for photon matching to cluster |
| Float_t m_RconeSep | 0.4 | Min cone R separation from clusters for photon isolation |
| Float_t m_RconeDep | 0.2 | R cone for energy deposition |
| Float_t m_MaxEdep | 10.0 | Max energy deposition for isolation |

Benchmark histograms for class ATLFPhotonMaker.

| ATLFPhotonMaker standard histograms | Title |
|---|---|
| TH1F *m_Mult | photon multiplicity |
| TH1F *m_MultHard | photon multiplicity hard |
| TH1F *m_MultHardIsol | photon multiplicity hard + isol |
| TH1F *m_E | (E-Ecru)/Ecru vers E photon ISOLATED |
| TH1F *m_Theta | (theta-thetacru) vers E photon ISOLATED |
| TH1F *m_Counter | counter vers E photon ISOLATED |
| TH1F *m_Mass2ph | pho-pho mass |

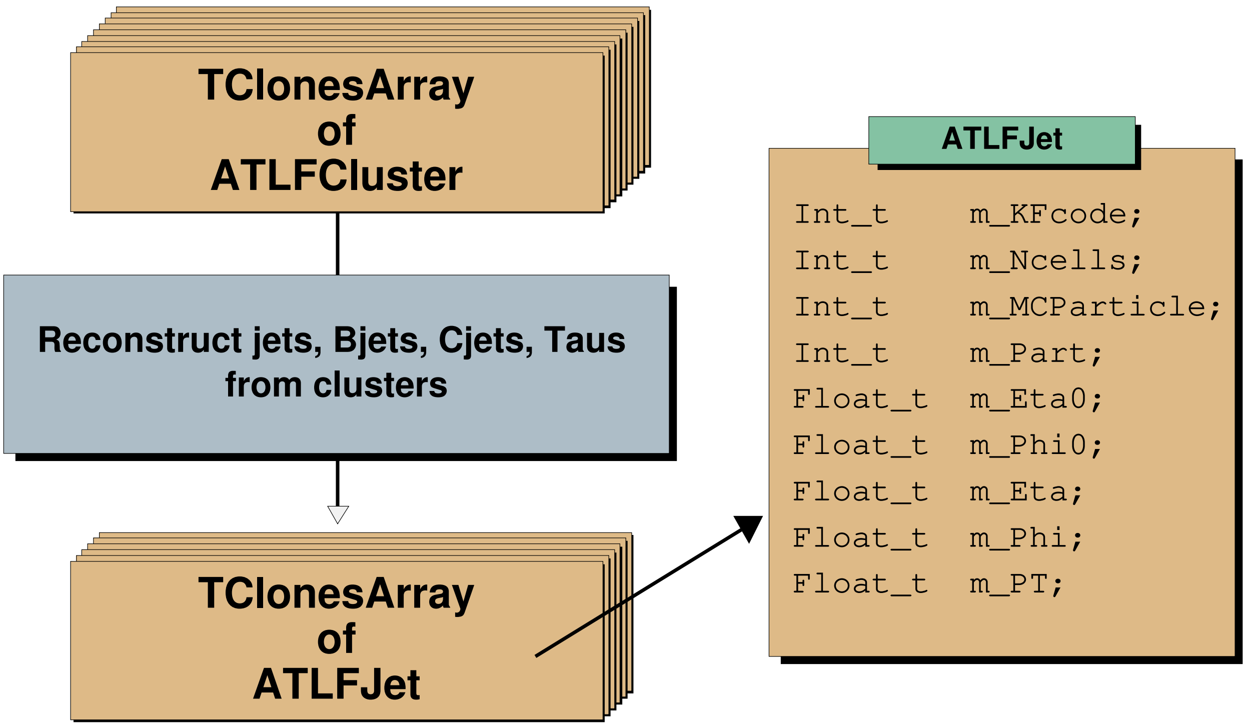
## ATLFJetMaker

Remaining clusters are smeared with parametrised resolutions and classified as jets. Labelling for b-jets, c-jets, τ-jets is performed.

The reconstructed electrons are stored as objects (**class ATLFJet**) into a **TClonesArray** list of jets.

Jets are labelled as:

- b-jets are stamped with **m_KFcode = 5**
- c-jets are stamped with **m_KFcode = 4**
- τ-jets are stamped with **m_KFcode = 15**
- other jets are stamped with **m_KFcode = 98**




**Example of loop on all reconstructed jets**

```
ATLFJetMaker *jetmaker = gATLFast->JetMaker();
TClonesArray *jets = jetmaker->Fruits();
TH1F *hPT = new TH1F("hPT","Transverse_Momentum",100,0,50);
ATLFJet *jet;
Int_t njets = jets->GetEntriesFast();
for (Int_t i=0;i<njets;i++) {
   jet = (ATLFJet*)jets->At(i);
   hPT->Fill(jet->PT());
}
```

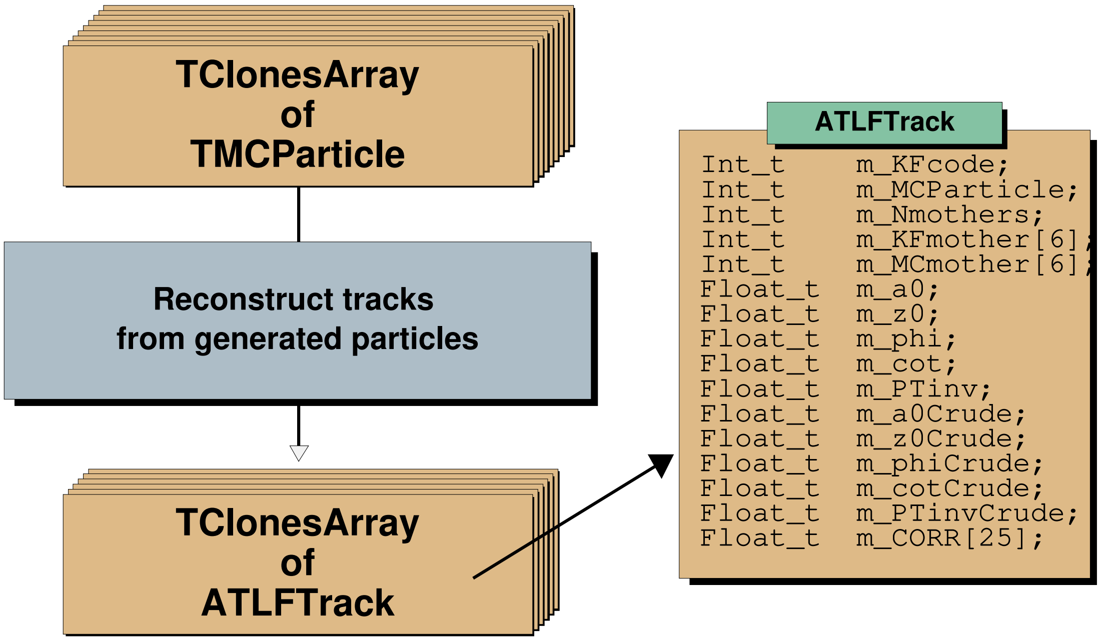
| ATLFJetMaker data members | Default value | Title |
|---|---|---|
| Int_t m_Njets | | Number of jets |
| Float_t m_MinET | 15.0 | Minimum PT for jets |
| Float_t m_BquarkMinPT | 5.0 | Rapidity coverage for jets |
| Float_t m_BquarkMaxEta | 5.0 | Minimum b-quark PT(after FSR) for b-jet matching |
| Float_t m_BRconeMatch | 2.5 | Maximum b-quark eta for b-jet matching |
| Float_t m_CquarkMinPT | 0.2 | Delta R for b-jet matching to b-quark |
| Float_t m_CquarkMaxEta | 5.0 | Minimum c-quark PT(after FSR) for c-jet matching |
| Float_t m_CRconeMatch | 2.5 | Maximum c-quark eta for c-jet matching |
| Float_t m_TMinPTHad | 0.2 | Delta R for c-jet matching to c-quark |
| Float_t m_TMaxEta | 10.0 | Minimum tau-had pt for tau-jet label |
| Float_t m_TRconeMatch | 2.5 | Maximum tau-eta for tau-jet label |
| Float_t m_TRconeMatch | 0.2 | Max R_tauj for tau-hadron matching |
| Float_t m_TRatioPT | 0.9 | Min ratio of tau-jet and hadron PT |

| Histograms | Title |
|---|---|
| TH1F *m_Mult | jet multiplicity |
| TH1F *m_Mass2jj | jetjet mass |
| TH1F *m_ERes4050 | energy resolution PT=40-50 GeV |
| TH1F *m_ERes200250 | energy resolution PT=200-250 GeV |
| TH1F *m_BMult | b-jets multiplicity |
| TH1F *m_BMultHard | b-quarks HARD multiplicity |
| TH1F *m_BPTFSR | PT b-jet/b-quark FSR |
| TH1F *m_BPTHard | PT b-jet/b-quark HARD |
| TH1F *m_BEtaFSR | eta b-jet/b-quark FSR |
| TH1F *m_BEtaHard | eta b-jet/b-quark HARD |
| TH1F *m_BPhiFSR | phi b-jet/b-quark FSR |
| TH1F *m_BPhiHard | phi b-jet/b-quark HARD |
| TH1F *m_BMjbjb | b-jets Mjbjb |
| TH1F *m_BMbbFSR | b-quark Mbb (after FSR) |
| TH1F *m_BMbb | M_bb b-jets/b-quark |

| ATLFJetMaker standard histograms | Title |
|---|---|
| TH1F *m_Mult | jet multiplicity |
| TH1F *m_Mass2jj | jetjet mass |
| TH1F *m_ERes4050 | energy resolution PT=40-50 GeV |
| TH1F *m_ERes200250 | energy resolution PT=200-250 GeV |
| TH1F *m_BMult | b-jets multiplicity |
| TH1F *m_BMultHard | b-quarks HARD multiplicity |
| TH1F *m_BPTFSR | PT b-jet/b-quark FSR |
| TH1F *m_BPTHard | PT b-jet/b-quark HARD |
| TH1F *m_BEtaFSR | eta b-jet/b-quark FSR |
| TH1F *m_BEtaHard | eta b-jet/b-quark HARD |
| TH1F *m_BPhiFSR | phi b-jet/b-quark FSR |
| TH1F *m_BPhiHard | phi b-jet/b-quark HARD |
| TH1F *m_BMjbjb | b-jets Mjbjb |
| TH1F *m_BMbbFSR | b-quark Mbb (after FSR) |
| TH1F *m_BMbb | M_bb b-jets/b-quark |
| TH1F *m_BRmin | Rmin b-jet, b-quark HARD |
| TH1F *m_CMult | c-jets multiplicity |
| TH1F *m_CMultHard | c-quarks HARD multiplicity |
| TH1F *m_CPTFSR | PT c-jet/c-quark FSR |
| TH1F *m_CPTHard | PT c-jet/c-quark HARD |
| TH1F *m_CEtaFSR | eta c-jet/c-quark FSR |
| TH1F *m_CEtaHard | eta c-jet/c-quark HAR |
| TH1F *m_CPhiFSR | phic-jet/c-quark FSR |
| TH1F *m_CPhiHard | phic-jet/c-quark HARD |
| TH1F *m_CMjcjc | c-jets Mjcjc |
| TH1F *m_CMccFSR | c-quark Mcc (after FSR) |
| TH1F *m_CMcc | M_cc c-jets/c-quark |
| TH1F *m_CRmin | Rmin c-jet, c-quark HAR |
| TH1F *m_TauJetMult | tau-jets multiplicity |
| TH1F *m_TausMult | taus multiplicity |

## ATLFTrackMaker

Starting from the list of generated particles, ATLFTrackMaker reconstructs helix parameters of charged particles. The reconstructed helix tracks parameters ($a_0$, $z_0$, $\phi$, $\cot(\Theta)$, $q/p_T$) are stored as objects (**class ATLFTrack**) into a **TClonesArray** list of tracks. Smearing is implemented using parametrized resolutions




**Example of loop on all reconstructed tracks**

```
ATLFTrackMaker *trackmaker = gATLFast->TrackMaker();
TClonesArray *tracks = trackmaker->Fruits();
TH1F *hPTinv = new TH1F("hPTinv","InversePT",100,0,50);
ATLFTrack *track;
Int_t ntracks = tracks->GetEntriesFast();
for (Int_t i=0;i<ntracks;i++) {
   track = (ATLFTrack*)tracks->At(i);
   hPTinv->Fill(track->PTinv());
}
```

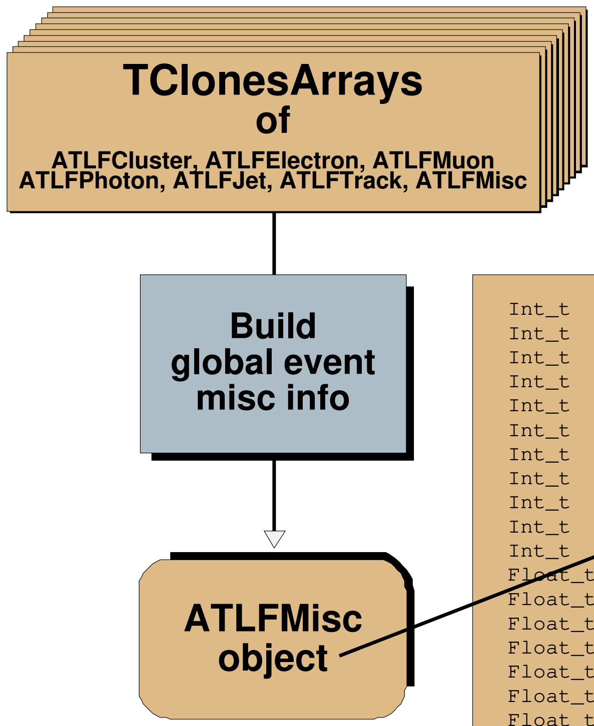
| **ATLFTrackMaker data members** | **Default value** | **Title** |
|---|---|---|
| Int_t m_Ntracks | | Number of tracks |
| F loat_t m_MinPT | 0.5 | Minimum PT for track |
| Float_t m_MaxEta | 2.5 | Maximum Eta for track |

| **ATLFTrackMaker standard histograms** | **Title** |
|---|---|
| TH1F *m_Mult | tracks multiplicity |

### ATLFMiscMaker

Calculates the missing transverse energy from the reconstructed transverse energy of identified electrons, muons, photons, reconstructed jets, remaining clusters and non-clustered cells.

The results are stored in one single object of **class ATLFMisc**. This object includes an event summary, the missing energy information and the number of reconstructed objects from all **Makers**.




**ATLFMisc**

```
Int_t   m_Run;          //Run number
Int_t   m_Event;        //Event number
Int_t   m_MCProcess;    //Scattering process code
Int_t   m_Nelectrons;   //Isolated electrons
Int_t   m_Nmuons;       //Isolated muons
Int_t   m_Nmuonsx;      //NonIsolated muons
Int_t   m_Nphotons;     //Isolated photons
Int_t   m_Nalljets;     //All jets
Int_t   m_Nbjets;       //b-labelled jets
Int_t   m_Ncjets;       //c-labelled jets
Int_t   m_Ntaujets;     //tau-labelled jets
Float_t m_CircJets;     //Circularity from jets
Float_t m_CircEvent;    //Circularity from cells
Float_t m_Thrust;       //Event Thrust
Float_t m_Oblateness;   //Event Oblateness
Float_t m_pxmiss;       //Missing momentum along X
Float_t m_pymiss;       //Missing momentum along Y
Float_t m_pxnu;         //Sum of missing X momenta
Float_t m_pynu;         //Sum of missing Y momenta
```

**Example of code to access global event info**

```
ATLFMiscMaker *miscmaker = gATLFast->MiscMaker();
ATLFMisc *misc = (ATLFMisc*)Fruit();
Float_t thrust = misc->Thrust();
```

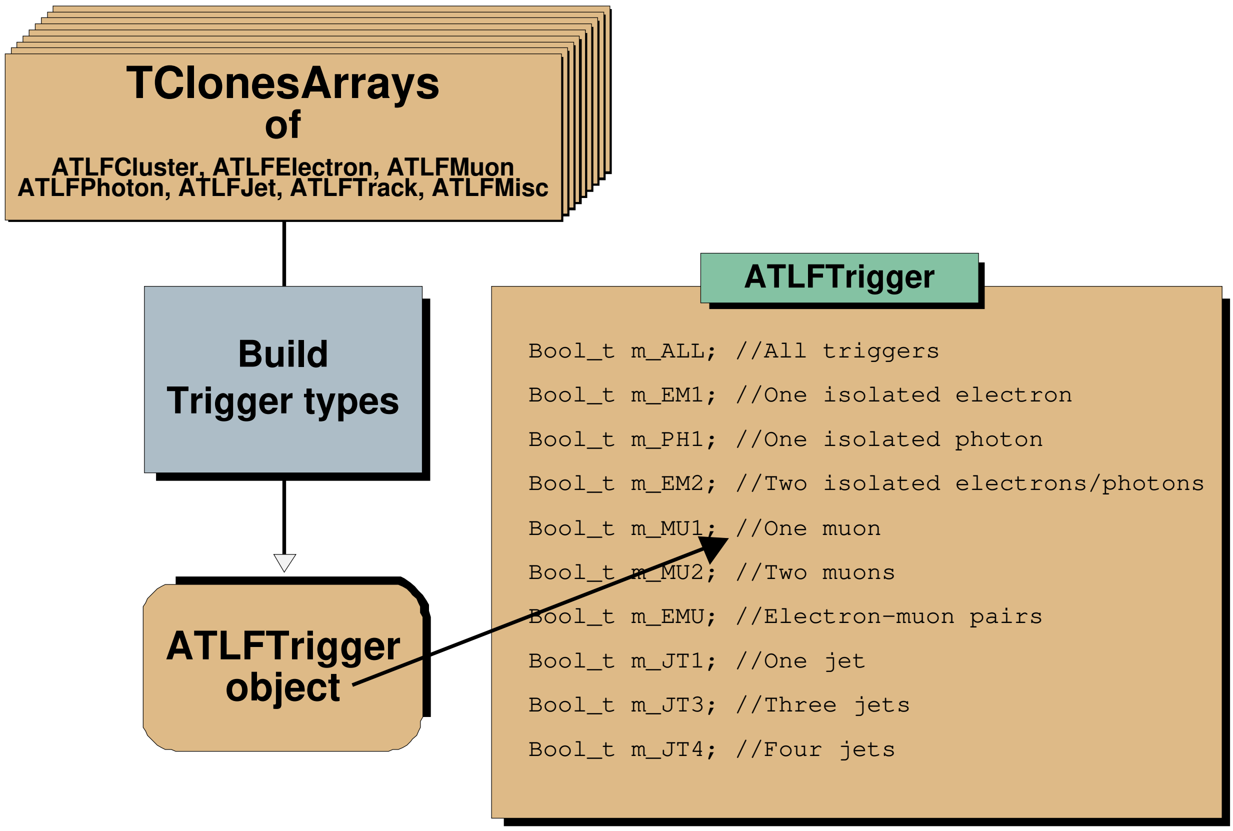
| ATLFMiscMaker data members | Default value | Title |
|---|---|---|
| Float_t m_MinET | 0.0 | Minimum E_T for unused cells |

| ATLFMiscMaker standard histograms | Title |
|---|---|
| TH1F *m_CircJets | circularity of jets |
| TH1F *m_CircEvent | circularity of event |
| TH1F *m_Thrust | thrust of event |
| TH1F *m_Oblateness | oblateness of event |
| TH1F *m_RecPT | reconstructed p_T |
| TH1F *m_RecPTcells | reconstructed p_T +cells |
| TH1F *m_PTnu | p_T nu |
| TH1F *m_PXnu | pX nu |
| TH1F *m_PYnu | pY nu |
| TH1F *m_RecPX | reconstructed pX |
| TH1F *m_RecPY | reconstructed pY |
| TH1F *m_RecPXcells | reconstructed pX + cells |
| TH1F *m_RecPYcells | reconstructed pY + cells |
| TH1F *m_ResolPX | resol missing pX |
| TH1F *m_ResolPX0 | resol missing pX (sumET=0-25) |
| TH1F *m_ResolPX25 | resol missing pX (sumET=25-50) |
| TH1F *m_ResolPX50 | resol missing pX (sumET=50-100) |
| TH1F *m_ResolPX100 | resol missing pX (sumET=100-200) |
| TH1F *m_ResolPX200 | resol missing pX (sumET=200-300) |
| TH1F *m_ResolPX300 | resol missing pX (sumET=300-400) |
| TH1F *m_PTmissPTnu | PTmiss/PTnu |
| TH1F *m_PXmissPXnu | PXmiss/PXnu |
| TH1F *m_PYmissPYnu | PYmiss/PYnu |

## ATLFTriggerMaker

Validates the selected physics event with a implified trigger procedure. Isolated electrons, photons, isolated and non-isolated muons and reconstructed jets are used for the trigger selection.

The results of all the trigger types are stored in one single object  of **class ATLFTrigger**.




### Example of code to access trigger info

```
ATLFTriggerMaker *triggermaker = gATLFast->TriggerMaker();
ATLFTrigger *trigger = (ATLFTrigger*)Fruit();
Bool_t all = trigger->ALL();
```

.

| ATLFTrigger data members | Default value | Title |
|---|---|---|
| Float_t m_EPhoEtaCoverage | 2.5 | eta-coverage for electrons/photons |
| Float_t m_EPhoMinPTlowEle | 20.0 | Minimum p_T for 1 electron low lumi |
| Float_t m_EPhoMinPThighEle | 30.0 | Minimum p_T for 1 electron high lumi |
| Float_t m_EPhoMinPTlowPho | 40.0 | Minimum p_T for 1 photon low lumi |
| Float_t m_EPhoMinPThighPho | 60.0 | Minimum p_T for 1 photon high lumi |
| Float_t m_EPhoMinPTlow2 | 15.0 | Minimum p_T for 2 ele/phot low lumi |
| Float_t m_EPhoMinPThigh2 | 20.0 | Minimum p_T for 2 ele/phot high lumi |
| Float_t m_MuoEtaCoverage | 2.4 | eta coverage for muons |
| Float_t m_MuoMinPTlowIso | 6.0 | Minimum p_T for 1 iso muon low lumi |
| Float_t m_MuoMinPThighIso | 20.0 | Minimum p_T for 1 iso muon high lumi |
| Float_t m_MuoMinPTlow | 15.0 | Minimum p_T for 1 muon low lumi |
| Float_t m_MuoMinPThigh | 30.0 | Minimum p_T for 1 muon high lumi |
| Float_t m_MuoMinPTlow2 | 6.0 | Minimum p_T for 2 muons low lumi |
| Float_t m_MuoMinPThigh2 | 10.0 | Minimum p_T for 2 muons high lumi |
| Float_t m_MuoMinPThigh2Iso | 6.0 | Minimum p_T for 2 iso muons high lumi |
| Float_t m_EMuoEtaCoverageE | 2.5 | eta-coverage for electrons |
| Float_t m_EMuoEtaCoverageM | 2.4 | eta-coverage for muons |
| Float_t m_EMuoMinPTlowEle | 5.0 | Minimum p_T for iso electron low lumi |
| Float_t m_EMuoMinPThighEle | 15.0 | Minimum p_T for iso electron high lumi |
| Float_t m_EMuoMinPTlowMuo | 6.0 | Minimum p_T for muon low lumi |
| Float_t m_EMuoMinPThighMuo | 6.0 | Minimum p_T for muon high lumi |
| Float_t m_JetEtaCoverage | 3.2 | eta coverage for jets |
| Float_t m_JetMinPTlow1jet | 300.0 | Minimum p_T for 1 jet low lumi |
| Float_t m_JetMinPThigh1jet | 300.0 | Minimum p_T for 1 jet high lumi |
| Float_t m_JetMinPTlow3jet | 80.0 | Minimum p_T for 3 jets low lumi |
| Float_t m_JetMinPThigh3jet | 100.0 | Minimum p_T for 3 jets high lumi |
| Float_t m_JetMinPTlow4jet | 30.0 | Minimum p_T for 4 jets low lumi |
| Float_t m_JetMinPThigh4jet | 50.0 | Minimum p_T for 4 jets high lumi |

## ATLFDisplay

This small class (about 600 lines) includes a 2-D and 3-D event display. It has been designed to display already precomputed events stored in the data base. The following functions are included;

- display of standard views (front,side,top,all)
- display of the various triggers
- 3-D display via the Root/**X3D** system. X3D is a very fast 3-D viewing facility, including wireframe, hidden line and hidden surface algorihms. A stereo view facility is also available. Once in the X3d canvas, type character M to see all the possibilities. X3D runs only on Unix systems and remains very efficient when used with the network (Xterminals,etc).
- 3-D display using **OPENGL**. This is fancier but slower and can barely be used with an Xterminal. Click on the HELP button to see all the possibilities.
- In all views (2-D and 3-D) a tube is displayed (radius=115cm, length = 640cm) to materialise the limit between central detector and calorimeters.
- The reconstructed objects are shown with tick lines. In the front view, the length of the line is proportional to the transverse energy of this object.
- You can click on any object with the right mouse button and select the **"Inspect"** menu item to display the object parameters.
- You can get a Postscript copy of the canvas by selecting the item **"SaveAs canvas.ps"** in the canvas **"File"** menu.

Further viewing parameters can be set by clicking in the edges of the canvas with the right mouse button. For example, one can change the PT cut to display particles, one can jump to any event number in the data base,etc.

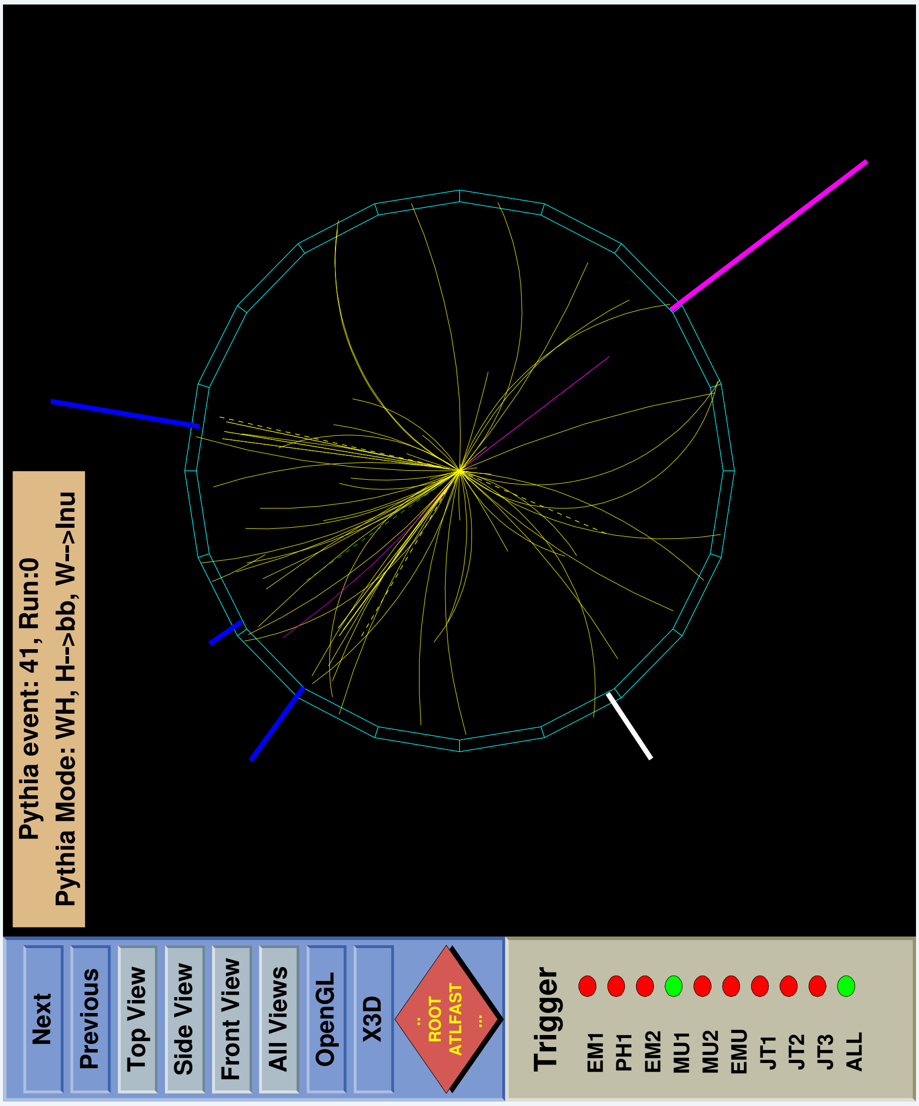
The following colors or line styles are used to display objects:

- photons are drawn with green dashed lines.
- neutral particles with yellow dashed lines
- electrons with red lines
- muons in cyan
- all other reconstructed particles in yellow. Non reconstructed particles in blue.
- reconstructed electrons with thick red lines
- reconstructed muons with thick cyan lines
- reconstructed photons with thick green lines
- reconstructed jets with thick dark blue lines
- unassigned clusters with thick light blue lines
- transverse missing energy with a tick white line

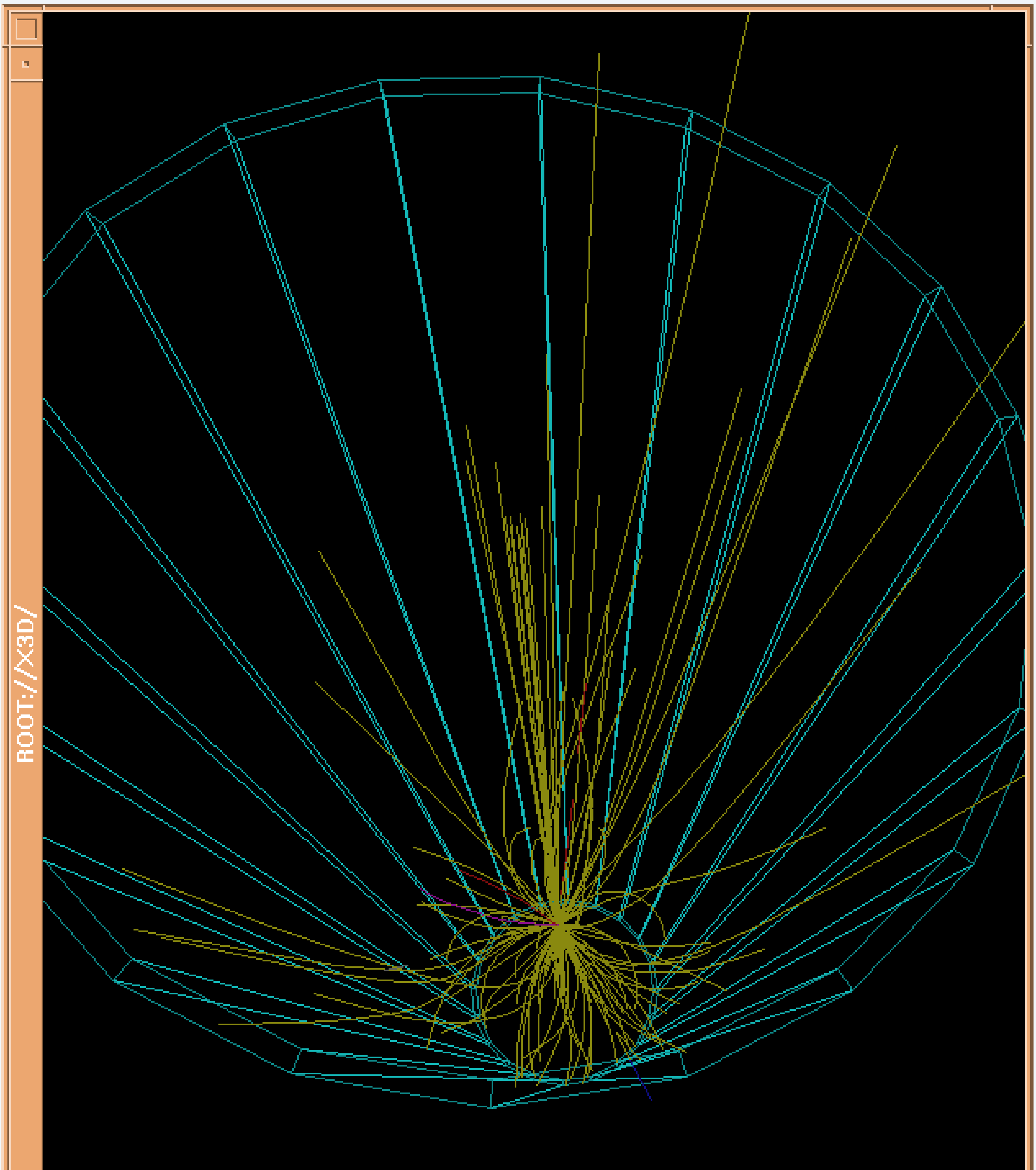
The event display can be started by executing the macro **display.C** below. This macro uses a file **"atlfastK.root"** generated by the macro **"umain.C"** executed with the statement **"gATLFast->MCMaker()->Save(2)"** in file **user.C**.

**macro display.C**

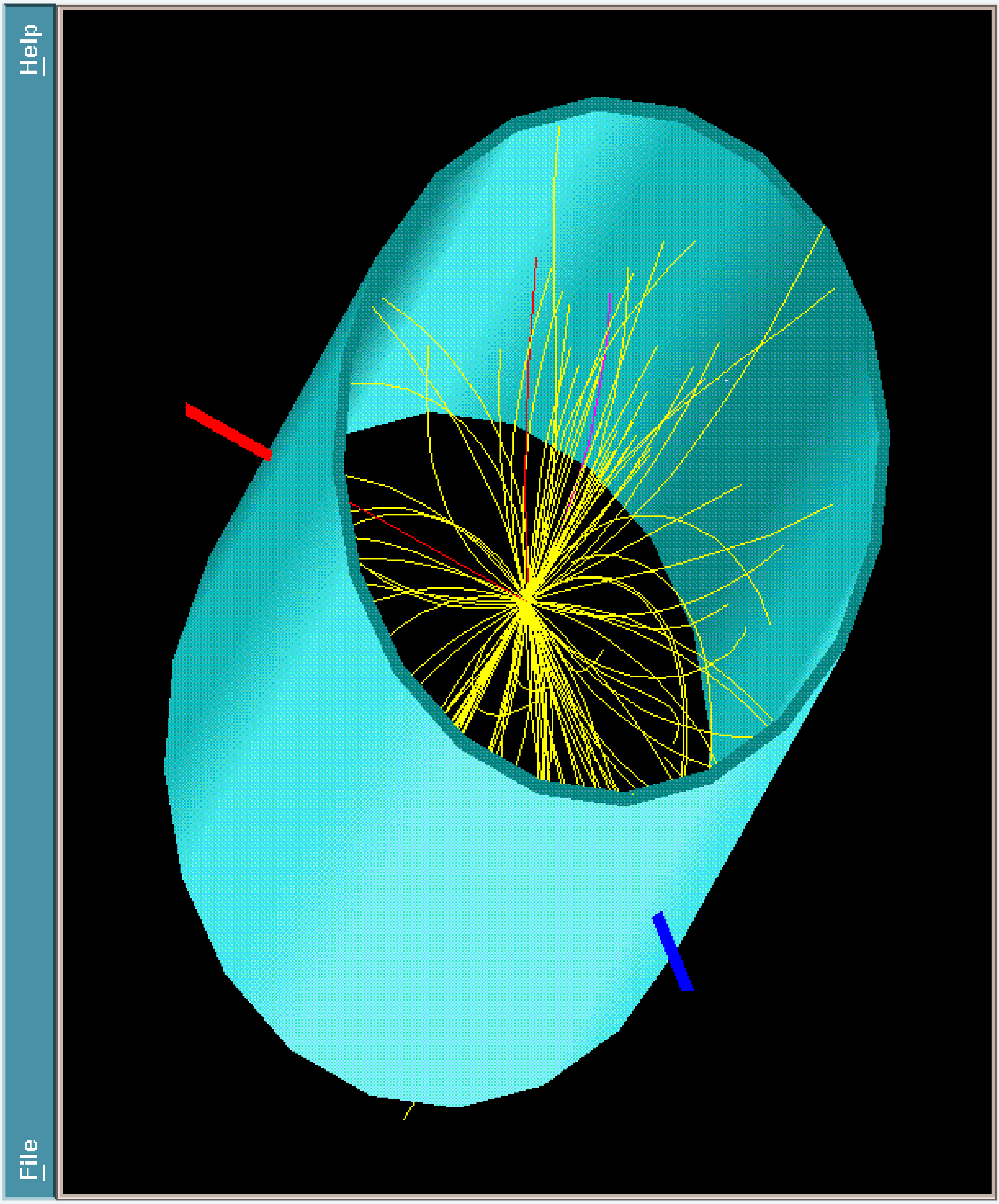
```
{
 gROOT->Reset();

 gROOT->Macro("LoadLibs.C");   // link standard Atlfast libs
 gROOT->Macro("GL.C");         // link OpenGL libs

 // Open the ATLFAST root event file
 TFile file("atlfastK.root");
 gATLFast = (ATLFast*)file.Get("atlfast");
 gATLFast->GetEvent(0);

 ATLFDisplay disp("ATLFast display");
 disp.ShowNextEvent(0);
}
```



# Wireframe view  with X3D



# OpenGL view of the same event



## The ROOT object browser

The browser can be used to quickly navigate the list of objects in memory or in a data base. A simple session connecting an Atlfast data base and invoking the browser is shown in the **macro "look.C"**

**macro look.C**

```
{
   gROOT->Reset();

   gROOT->Macro("$ATLFAST/loadlibs");
   gROOT->Macro("GL.C");

   // Open the root output file
   TFile file("atlfast.root");
   ATLFast *atlfast = (ATLFast*)file.Get("atlfast");
   atlfast->GetEvent(0);

   // Invoke the object browser
   TBrowser b;
}
```

The browser shows two panes: the containers pane and the objects pane. In the containers pane (left), you see the list of browsable containers. Click on the **ATLFAST container to see** new containers:

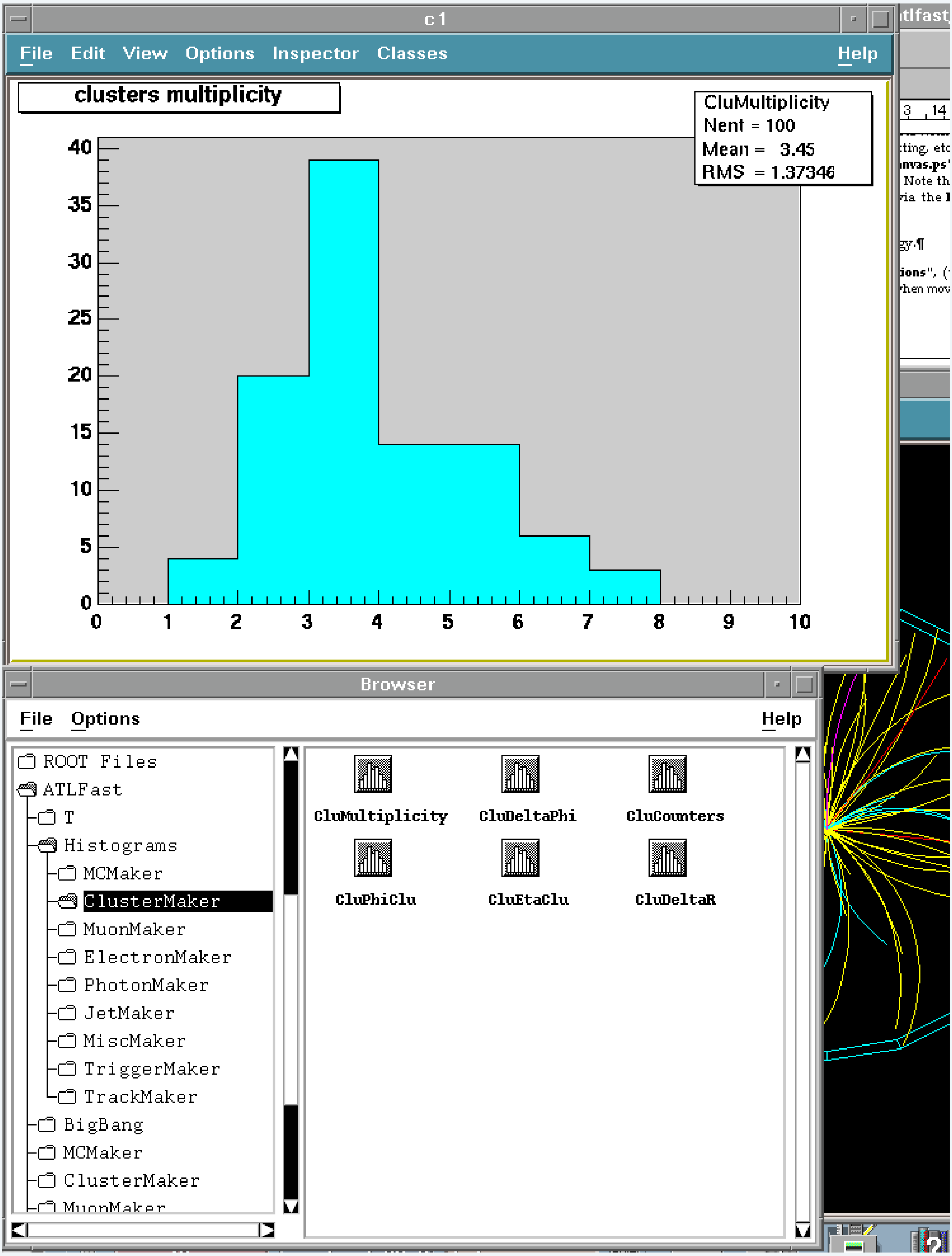
- The Tree "T"
- The histograms containers
- The BigBang container.
- The Makers containers

Clicking on the **Tree "T"** shows the corresponding **Maker branches**. For example, selecting **"Clusters"**, will display in the right pane all the attributes of **ATLFCluster**. Clicking on one attribute will automatically loop on all events, for each event on all clusters, and then histogram this attribute.

Clicking on the **"Histograms"** container displays a similar organisation. Selecting ClusterMaker will show in the right pane the list of standard histograms for the **ClusterMaker**. Click on one histogram to draw it. Using the right mouse button instead, gives a choice of different functions (fit,etc). Once an histogram is displayed in the graphics canvas, you can point to the histogram contour with the right mouse button and select one of the proposed actions (changing histogram attributes, fitting, etc.). You can produce a Postscript file of the picture in the canvas by selecting the item **"Save as canvas.ps"** in the canvas **"File"** menu. See the **"HELP"** buttons in the browser or canvas for more information. Note that before drawing an histogram, you should create a new canvas. You can create a new canvas via the **File** menu in the browser or event display canvas

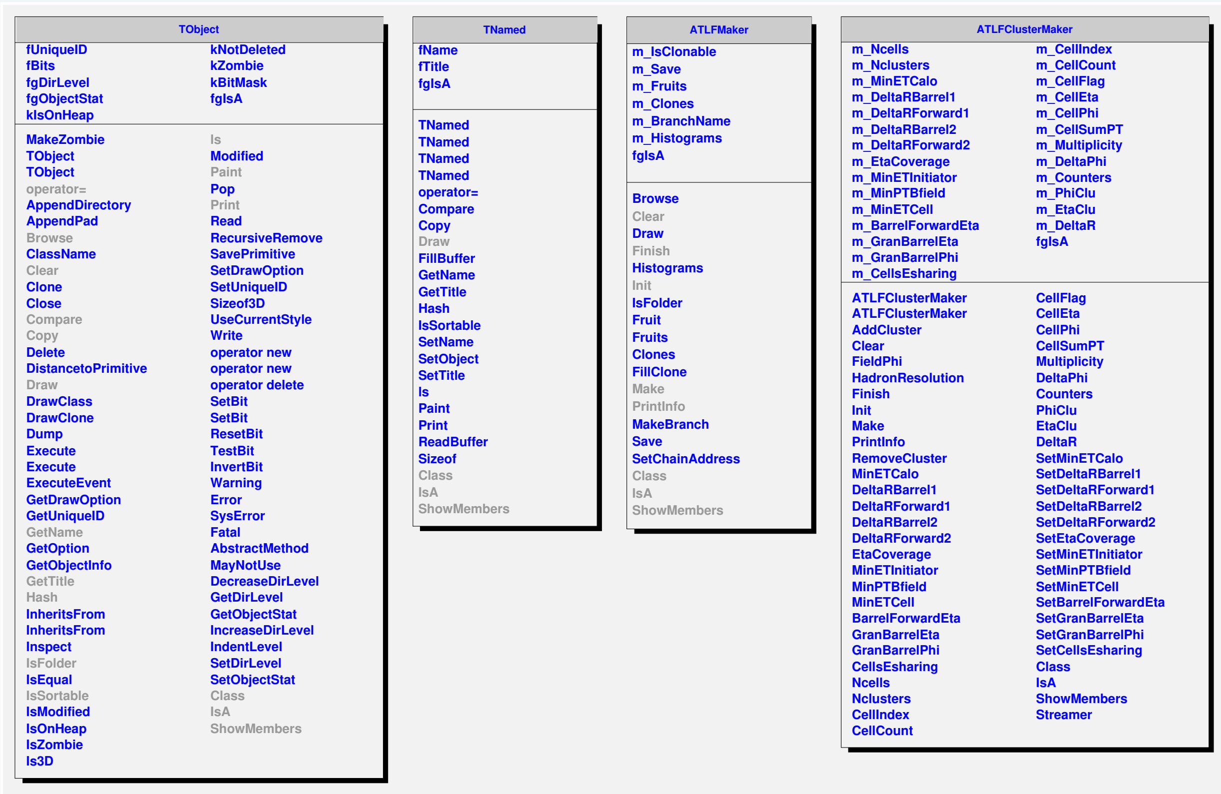
Clicking on the **BigBang** container shows the primary collision genealogy.

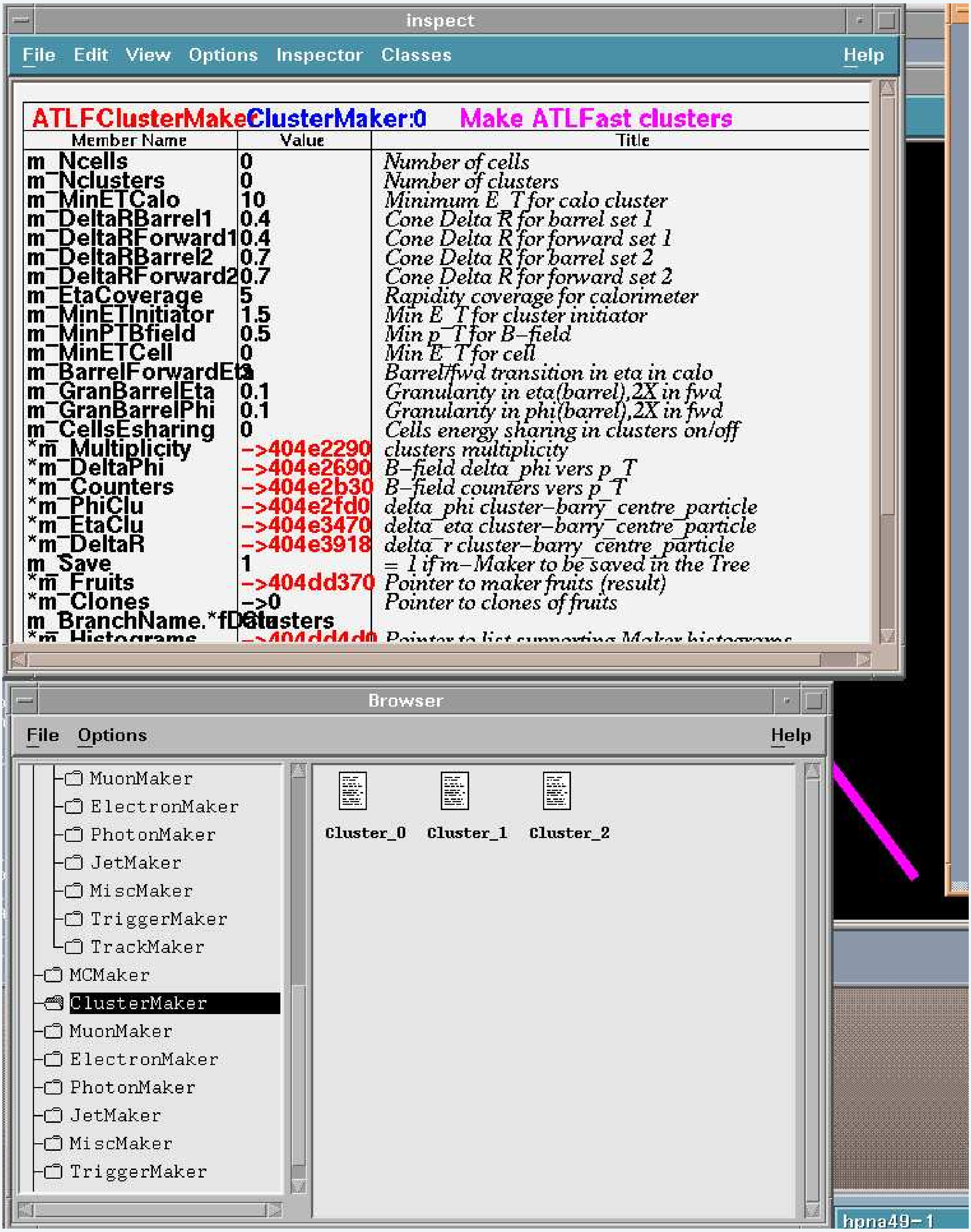
If you activate the item **"Show Event Status"** in the canvas menu **"Options"**, (try this feature in the event display canvas), important attributes of an object will be displayed when moving the mouse on top of the object in the canvas


c1
File Edit View Options Inspector Classes Help
clusters multiplicity
CluMultiplicity
Nent = 100
Mean = 3.45
RMS = 1.37346
40
35
30
25
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9 10
Browser
File Options Help
ROOT Files
ATLFast
T
Histograms
MCMaker
ClusterMaker
MuonMaker
ElectronMaker
PhotonMaker
JetMaker
MiscMaker
TriggerMaker
TrackMaker
BigBang
MCMaker
ClusterMaker
CluMultiplicity
CluDeltaPhi
CluCounters
CluPhiClu
CluEtaClu
CluDeltaR

## The ROOT object inspector

Any object derived from the **Root class TObject** can be inspected. For example, click one one track in the event display with the right mouse button and select the **"Inspect"** action. You will get a new window with the dump of the corresponding object (data member name, value and title). If the data member is a pointer to another object, you can click on the pointer to inspect the pointed object,etc. In the same way, selecting the **"DrawClass"** item in the context menu will display the class inheritance tree in a separate window. see below.




inspect
File Edit View Options Inspector Classes Help
ATLFClusterMaker ClusterMaker:0 Make ATLFast clusters
Member Name Value Title
m_Ncells 0 Number of cells
m_Nclusters 0 Number of clusters
m_MinETCalo 10 Minimum E_T for calo cluster
m_DeltaRBarrel1 0.4 Cone Delta R for barrel set 1
m_DeltaRForward1 0.4 Cone Delta R for forward set 1
m_DeltaRBarrel2 0.7 Cone Delta R for barrel set 2
m_DeltaRForward2 0.7 Cone Delta R for forward set 2
m_EtaCoverage 5 Rapidity coverage for calorimeter
m_MinETInitiator 1.5 Min E_T for cluster initiator
m_MinPTBfield 0.5 Min p_T for B-field
m_MinETCell 0 Min E_T for cell
m_BarrelForwardEta 3 Barrel/fwd transition in eta in calo
m_GranBarrelEta 0.1 Granularity in eta(barrel),2X in fwd
m_GranBarrelPhi 0.1 Granularity in phi(barrel),2X in fwd
m_CellsEsharing 0 Cells energy sharing in clusters on/off
*m_Multiplicity ->404e2290 clusters multiplicity
*m_DeltaPhi ->404e2690 B-field delta_phi vers p_T
*m_Counters ->404e2b30 B-field counters vers p_T
*m_PhiClu ->404e2fd0 delta_phi cluster-barry_centre_particle
*m_EtaClu ->404e3470 delta_eta cluster-barry_centre_particle
*m_DeltaR ->404e3918 delta_r cluster-barry_centre_particle
m_Save 1 = 1 if m-Maker to be saved in the Tree
*m_Fruits ->404dd370 Pointer to maker fruits (result)
*m_Clones ->0 Pointer to clones of fruits
m_BranchName.*fData Clusters
Browser
File Options Help
MuonMaker
ElectronMaker
PhotonMaker
JetMaker
MiscMaker
TriggerMaker
TrackMaker
MCMaker
ClusterMaker
MuonMaker
ElectronMaker
PhotonMaker
JetMaker
MiscMaker
TriggerMaker
Cluster_0
Cluster_1
Cluster_2
hpna49-1

## Class ATLFastB

This class is supposed to be used on the **Tree** produced by the **ATLFast** class to simulate tagging/identification efficiencies in a parametrised way.

The following methods are provided

- **ATLFastB::CalibJet()** for jets energy calibration
- **ATLFastB::TagBjet()** for b-tagging simulation
- **ATLFastB::TagTaujet()** for tau-tagging simulation
- **ATLFastB::TauRejet()** for tau-veto simulation

| ATLFastB data memebers | Default value | Title |
|---|---|---|
| Int_t m_BtagSet | 5 | Option for b-tagging (nset=1,2,3,4,5) |
| Float_t m_TautagEpsi | 0.5 | Efifciency for tau-tagging (epsi=0.1-0.9) |
| Float_t m_TauVetoSet | 2 | Veto set for non-tau jets |

The following set of benchmark histograms is provided. ;

| ATLFast standard histograms | Title |
|---|---|
| TH1F *m_hist11 | Labelled jet flavour |
| TH1F *m_hist12 | Tagged jet flavour |
| TH1F *m_hist13 | Jets pT before calibration |
| TH1F *m_hist14 | Jets pT after calibration |

## Examples:

- **UserAnal++**

  This simple example shows how to generate PYTHIA events, browse content of the ROOT Tree and display events.

  - The macro **UserAnal_main.C** generates file **atlfast.root**. The user routine **UserAnal. C** shows how to set the generation parameters **User::SetRunParameters()**, change detector/reconstruction related parameters **SetMakerParameters(),** have a filter for event storage **User::Filter(Int_t &kFilter)**, fill histograms on-fly with simulation **User::FillHistograms()**, store histograms **User::StoreHistograms()** and produce **.eps** file. The stored Tree can be browsed with the macro **User_look.C**

  - The macro **UserAnal_mainK.C** generates file **atlfastK.root** which in addition stores all particles from LUJET common blocks. The **event display** of these events can be obtained with the macro **User_display.C**.

- **WMassAnal++**

  This is a more advanced example of ageneration and interactive analysis.

  - **WMassGen_main.C** generates file **WMass_events.root** with W-->jj and W-->l$\nu$ events.

  - **WMassGen.C** defines a user class which sets the generation filter and control histograms (as in the example above)

  - **WMassAnal_main.C** analyses the file **WMass_events.root** and produces a file **WMass_anal.root** with histograms resulting from the analysis. **WMassAnal.C** defines class for above analysis: reconstruction of W mass in both decay modes. **WMassAnal::AnalMjj(TH1F *hist)** reconstructs dijet decay mode and **WMassAnal::AnalMln(TH1F *hist)** reconstructs semileptonic decay mode. **WMassAnal::Finish()** stores histograms.

  - The **ATLFastB** classes are also used; in this case **ATLFastB::Calibjet()** for jets recalibration. It is organised by the macro **WMassAnal_main.C** , see next page.

- **TauMass++**

  This example shows the **A-->**$\tau\tau$ analysis. It contains also a few examples to generate a nice **.eps** file.

- **TopMass++**

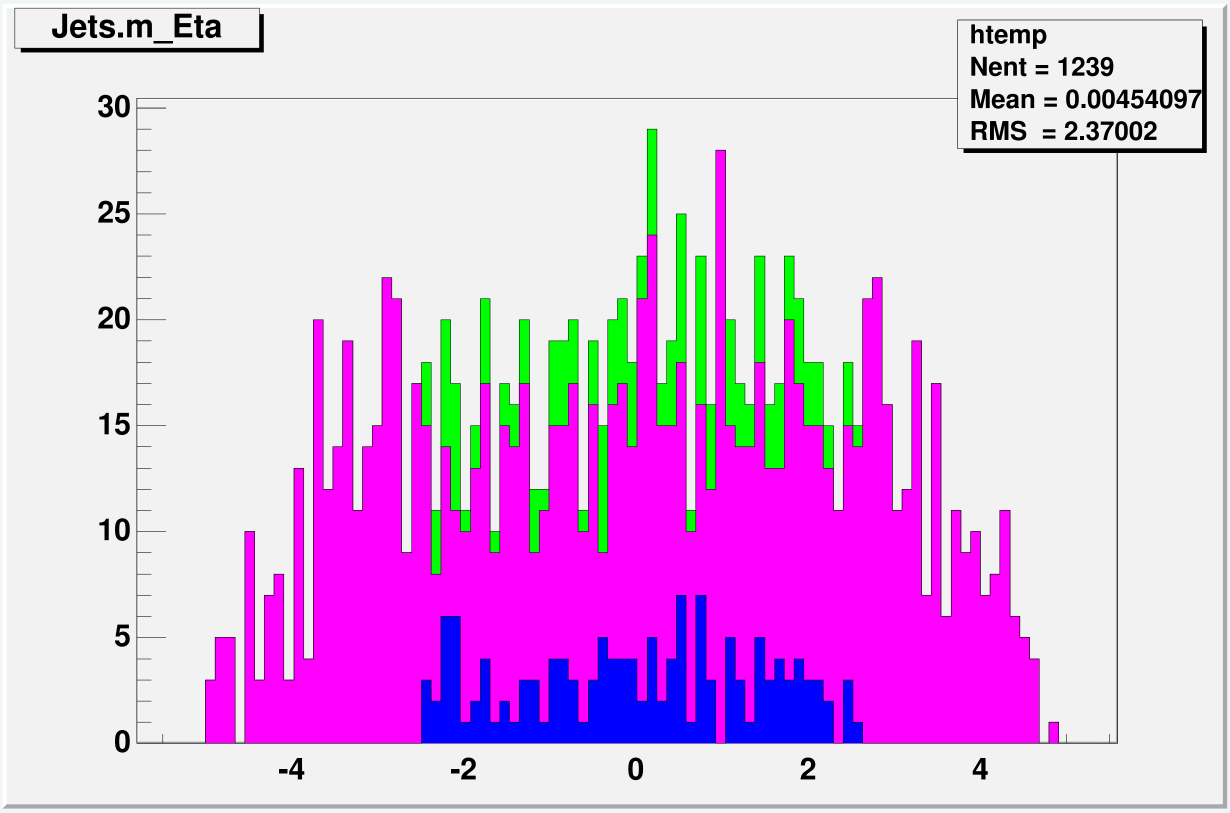
  This is the most advanced example of production and analysis which is run in batch. The work is organised in subdirectories for generation **TopMass++/Gen**, analysis **TopMass++/Anal** and ROOT Tree storage **TopMass++/Ntup**. For each subdirectory **makefiles** and **README** are available. Here the **ATLFastB class** is used for calibration **ATLFastB::Calibjet()** and b-jet tagging **ATLFastB::Tagbjet()**. In **TopMass++/Anal/TopMassAnal_main.C** you will find an example how to analyse several **atlfast.root** files in a **TChain**.

### Example WMassAnal_main.C

```
//   example of main program/macro to run analysis
void WMassAnal_main(Long_t nevmax=-1)
{
   gROOT->Reset();

   gROOT->Macro("loadlibs");

   // Open the root output file
   TFile *file = new TFile("WMass_events.root");

   // create pointer to objet atlfast
   ATLFast *atlfast = (ATLFast*)file->Get("atlfast");

  // create pointer to objet detect
   ATLFastB *atlfastb = new ATLFastB();
   atlfastb->Init();

   // create pointer to objet wmass
   gROOT->LoadMacro("WMassAnal.C");  // execute WMass macro to
compile/load
                                     // WMassAnal class.
   WMassAnal *wmass = new WMassAnal();  // create WMassAnal object
   wmass->Init();

   // create pointers to objet tree
   TTree *tree    = atlfast.Tree();

   Long_t nevents = tree->GetEntries();
   if (nevmax > 0) nevents = nevmax;
   Long_t ev;
   for (ev=0; ev<nevents; ev++) {
      if (ev%100 == 1) printf("In loop:%d of %d\n",ev,nevents);
      tree->GetEvent(ev);         // Get reconstructed event
      atlfastb->CalibJet();      // Recalibrate reconstructed jets
      wmass->Analyse();          // Do your analysis
      atlfast->Clear();          // Clear all event lists
   }
   atlfastb->Finish();
   wmass->Finish();
}
```

## Example of interactive session.

```
//      Example of a small session with cuts at the command line
   gROOT->Reset();
   TFile f("atlfast.root");

//   make a first pass with no cuts on Eta of all jets
   T.Draw("Jets.m_Eta");

//   new pass selecting non bjets, Cjets or Taujets
   T.Draw("Jets.m_Eta","Jets.m_KFcode == 98","same");

//   last pass with only Bjets
   T.Draw("Jets.m_Eta","Jets.m_KFcode == 5","same");

//  We have the 3 plots on the screen.
//  Use the context menu to set colors
```



## Timing and Space

The total time to run the Fortran version or C++ version are equivalent. ATLFast in C++ is about 10 to 20 per cent faster than the corresponding Fortran version. The Pythia event generator part is a bit slower because one has to import the data in the /LUJETS/ common block into objects. Fot 10000 events and an average of 600 particles per event each particle has 15 attributes (60 bytes), one has to copy 360 Mbytes !!. Because the Root/C++ version provides support for data compression, we decided to save much more information per event into the data base (about a factor 4 more).

| **10 000 events** | **PYTHIA** | **ATLFAST** |
|---|---|---|
| FORTRAN | | with I/O (no I/O) |
| $H \rightarrow \gamma\gamma$ | 1017 sec | 277 sec (264 sec) |
| $Z \rightarrow e^+e^-$ | 853 sec | 212 sec (208 sec) |
| $Z \rightarrow \mu^+\mu^-$ | 756 sec | 174 sec (170 sec) |
| WH , $H \rightarrow b\bar{b}$ | 890 | 322 sec (291 sec) |
| OO/C++ | | |
| $H \rightarrow \gamma\gamma$ | 1085 sec | 264 sec (199 sec) |
| $Z \rightarrow e^+e^-$ | 813 sec | 219 sec (167 sec) |
| $Z \rightarrow \mu^+\mu^-$ | 809 sec | 192 sec (169 sec) |
| WH , $H \rightarrow b\bar{b}$ | 973 sec | 320 sec (240 sec) |

The Table below shows the data base size for various physics processes. The column **"CWN ntuple"** shows the size of the PAW ntuple for the Fortran version. The column **"Root Tree (arrays)"** shows the size of this PAW ntuple when automatically converted by the Root utility called **"h2root"**. The last column **"Root Tree (objects)"** shows the size of the Root data base containing the full C++ lists of objects produced by each Maker. If compression is off (an option when producing the data base), the size would be 24.6 Mbytes instead of 3.3 with the compression option.

| **10 000 events** | **CWN Ntyple** | **ROOT Tree (arrays)** | **ROOT Tree (Objects) 4X more info** |
|---|---|---|---|
| $H \rightarrow \gamma\gamma$ | 5.5 MB | 2.7 MB | 3.3 MB (no comp. 24.6 MB) |
| $Z \rightarrow e^+e^-$ | 4.9 MB | 2.5 MB | 2.9 MB |
| $Z \rightarrow \mu^+\mu^-$ | 5.2 MB | 2.3 MB | 2.8 MB |
| WH , $H \rightarrow b\bar{b}$ | 6.0 MB | 3.4 MB | 4.0 MB |
| $t\bar{t} \rightarrow$ jjblvb | 6.0 MB | 4.0 MB | 5.2 MB |

## Installation procedure

- **Importing ROOT libraries and executables**
  - on the Web, go to URL http://root.cern.ch/root/Version103.html
  - click on the machine of your choice. This will automatically transfer a compressed tar file. Copy the file to your home directory.
  - gtar zxvf rootxxxxx.tar.gz.

This will create a directory called root and subdirectories

  - root/bin with the executables root, h2root and g2root
  - root/lib with all root shared libraries
  - root/include with all the root include files
  - root/tutorials about 35 example macros
  - root/test about 12 batch test examples
  - create a symbol **$ROOTSYS** pointing to the root directory.
  - Set your **LD_LIBRARY_PATH**

(see examples in http://root.cern.ch/root/Availability.html)

- **Importing the Pythia libraries**
  - Create a directory $HOME/pythia
  - Create a symbol **$PYTHIA** pointing to this pythia directory.
  - cd $HOME
  - ftp the pythia tar file from hpbrun.cern.ch from public/pythia
  - gtar zxvf pythia.tar.gz
- **Importing ATLFAST**
  - Create a directory $HOME/atlfast
  - Create a symbol **$ATLFAST** pointing to this atlfast directory.
  - ftp the atlfast tar file from hpbrun.cern.ch from public/atlfast/atlfast.tar.gz
  - cd $HOME
  - tar zxvf atlfast.tar.gz
  - cd $ATLFAST
  - Create the two following links
    - ln -s LoadLibs.hpux loadlibs (if on hpux,etc..)
    - ln -s Makefile.hpux makefile
  - gmake clean
  - gmake depend
  - gmake

3

- - **Importing ATLFAST-B**
    - Create a directory $HOME/atlfastB-oo
    - Create a symbol $ATLFASTB pointing to this atlfastb directory.
    - cd $HOME
    - ftp the atlfastB-oo tar file ~erichter/public/ATLFAST-dir/dev++
    - gtar zxvf atlfastB-oo.tar.gz
    - gmake
- **Importing Examples**
    - cd $HOME
    - ftp the examples tar file ~erichter/public/ATLFAST-dir/dev++
    - to start TopMassAnal++ do:
        - cd TopMassAnal++
        - gmake

At this point, you should have all the material needed.

Before starting, modify your search path to include **$ROOTSYS/bin**.

Insert the definition of the **$ROOTSYS, $PYTHIA, $ATLFAST, $ATLFASTB** in your startup script.

# Class TPythia

**TPythia class** organises access to T. Sjostrand's PYTHIA 5.7/Jetset 7.4 event generator which is written in FORTRAN. **TPythia object** is something which can be called an "interface" between C++/ROOT and FORTRAN. It copies all particles generated by PYTHIA into its own list of **TMCParticle objects** (each TMCParticle object describes one entry in PYTHIA's LUJETS common). One can access all settings in control commons using **TPythia's methods** (i.e. FORTRAN statement **'MSTP=0'** will be replaced by **pythia->SetMSTP(0)**;' and the value of MSTP might be get using '**pythia->GetMSTP();** ).

The TPythia public data members

- **Pythia and Lujets subroutines access**

  virtual void **PyStat(Int_t mstat=1);**

  virtual void **PyTest(Int_t key=1);**

  virtual void **PyEvnt();**

  virtual void **LuList(Int_t mlist=1);**

  virtual void **LuExec();**

  virtual Int_t **LuComp(Int_t kf);**

  virtual void **PyInit(char *frame, char *beam, char *target, float win);**

- **Generic functions**

  virtual void **SetMSEL(Int_t sel);**

  virtual Int_t **GetMSEL() const;**

  virtual void **SetMSUB(Int_t isub, Bool_t msub=1);**

  virtual Bool_t **GetMSUB(Int_t isub) const;**

  virtual void **SetKFIN(Int_t i, Int_t j, Bool_t kfin=1);**

  virtual Bool_t **GetKFIN(Int_t i, Int_t j) const;**

  virtual void **SetCKIN(Int_t key, Float_t value);**

  virtual Float_t **GetCKIN(Int_t key) const;**

- **Common /PYPARS/ access routines**

  virtual void **SetMSTP(Int_t key, Int_t value);**

  virtual Int_t **GetMSTP(Int_t key) const;**

  virtual void **SetPARP(Int_t key, Float_t value);**

  virtual Float_t **GetPARP(Int_t key) const;**

  virtual void **SetMSTI(Int_t key, Int_t value);**

  virtual Int_t **GetMSTI(Int_t key) const;**

  virtual void **SetPARI(Int_t key, Float_t value);**

  virtual Float_t **GetPARI(Int_t key) const;**

- **Common /PYINT1 / access routines**

```
virtual void      SetMINT(Int_t key, Int_t value);
virtual Int_t     GetMINT(Int_t key) const;
virtual void      SetVINT(Int_t key, Float_t value);
virtual Float_t   GetVINT(Int_t key) const;
virtual void      SetICOL(Int_t kf,Int_t i, Int_t j,Int_t value);
virtual Int_t     GetICOL(Int_t kf,Int_t i, Int_t j) const;
```

- **Common PYINT2 access routines:**

```
virtual void      SetISET(Int_t isub,Int_t iset);
virtual Int_t     GetISET(Int_t isub) const;
virtual void      SetKFPR(Int_t isub, Int_t j, Int_t kfpr);
virtual Int_t     GetKFPR(Int_t isub, Int_t j) const;
virtual void      SetCOEF(Int_t isub, Int_t j, Float_t coef);
virtual Float_t   GetCOEF(Int_t isub, Int_t j) const;
virtual void      SetICOL(Int_t kf,Int_t i, Int_t j,Int_t value);
virtual Int_t     GetICOL(Int_t kf,Int_t i, Int_t j) const;
```

- **Common PYINT3 access routines - "read only" !!!**

```
virtual Float_t   GetXSFX(Int_t side, Int_t kf) const;
virtual Int_t     GetISIG(Int_t ichn, Int_t isig) const;
virtual Float_t   GetSIGH(Int_t ichn) const;
```

- **Common PYINT4 access routines - "read only" !!!**

```
virtual Float_t   GetWIDP(Int_t kf, Int_t j) const;
virtual Float_t   GetWIDE(Int_t kf, Int_t j) const;
virtual Float_t   GetWIDS(Int_t kf, Int_t j) const;
```

- **Common PYINT5 access routines - "read only" !!!**

```
virtual Int_t     GetNGEN(Int_t isub, Int_t key) const;
virtual Float_t   GetXSEC(Int_t isub, Int_t key) const;
```

- **Common /LUDAT1 / access:**

```
virtual Int_t     GetMSTU(Int_t key) const;
virtual void      SetMSTU(Int_t key, Int_t value);
virtual Float_t   GetPARU(Int_t key) const;
virtual void      SetPARU(Int_t key, Float_t value);
virtual Int_t     GetMSTJ(Int_t key) const;
virtual void      SetMSTJ(Int_t key, Int_t value);
virtual Float_t   GetPARJ(Int_t key) const;
virtual void      SetPARJ(Int_t key, Float_t value);
```

- **Common /LUDAT2 / access:**

  virtual Int_t **GetKCHG(Int_t kc, Int_t key) const;**

  virtual void **SetKCHG(Int_t kc, Int_t key, Int_t value);**

  virtual Float_t **GetPMAS(Int_t kc, Int_t key) const;**

  virtual void **SetPMAS(Int_t kc, Int_t key, Float_t value);**

  virtual Float_t **GetPARF(Int_t key) const;**

  virtual void **SetPARF(Int_t key, Float_t value);**

  virtual Float_t **GetVCKM(Int_t i, Int_t j) const;**

  virtual void **SetVCKM(Int_t i, Int_t j, Float_t value);**

- **Common /LUDAT3/ access:**

  virtual Int_t **GetMDCY(Int_t kc, Int_t key) const;**

  virtual void **SetMDCY(Int_t kc, Int_t key, Int_t value);**

  virtual Int_t **GetMDME(Int_t idc, Int_t key) const;**

  virtual void **SetMDME(Int_t idc, Int_t key, Int_t value);**

  virtual Float_t **GetBRAT(Int_t idc) const;**

  virtual void **SetBRAT(Int_t idc, Float_t value);**

  virtual Int_t **GetKFDP(Int_t idc, Int_t j) const;**

  virtual void **SetKFDP(Int_t idc, Int_t j, Int_t value);**

- **Common /LUDAT4/ access:**

  virtual Text_t **GetCHAF(Int_t kc) const;**

  virtual void **SetCHAF(Int_t kc,Text_t *value);**

- **Common /LUDATR / access:**

  virtual void **SetMRLU(Int_t key, Int_t seed);**

  virtual Int_t **GetMRLU(Int_t key) const;**

  virtual void **SetRRLU(Int_t key, Float_t value);**

  virtual Float_t **GetRRLU(Int_t key) const;**

- **Other Methods:**

  A few other useful methods is provided . They are direct interfaces to FORTRAN PYTHIA/LUJET's subroutines, and has the same functionality:

  **void TPythia::PyStat(Int_t mstat=1); ====> PYSTAT**

  **void TPythia::PyTest(Int_t key=1); ====> PYTEST**

  **void TPythia::PyEvnt(); ====> PYEVNT**

  **void TPythia::LuList(Int_t mlist=1); ====> LULIST**

  **void TPythia::LuExec(); ====> LUEXEC**

  **Int_t TPythia::LuComp(Int_t kf); ====> LUCOMP**

# ROOT Dependencies

The current implementation of **ATLFAST++ is based on the ROOT framework**. It uses standard C++, but no templates. The experience in the development of the ROOT system strongly suggests to avoid templates for portability reasons and also to avoid unnecessary large binaries. The implementation is strongly object-oriented, all objects derive from the base **class TObject**. This simple structure guarantees for example an easy conversion or adaptation to Java.

- **ATLFast class**

  uses the container **class TList** (a linked list) to store the list of Makers. All class libraries on the market provide the equivalent. The adaptation is straightforward.

- **ATLFMaker**

  This is the base class for all Makers. It derives from **TNamed/TObject**. Derivation from **TNamed** is straigthforward to remove. The iteration on all Makers assumes only the type **ATLFMaker**. Removing **TNamed**, you loose the possibility to inspect objects. The main dependency comes from the use of the container **class TClonesArray** (see discussion below). The **TClonesArray** is a special array of objects designed to store a large quantity of objects of the same type in a loop of events. **TClonesArray** can be replaced by a "standard" variable array of objects. ROOT itself has such a class called **TObjArray**. However, moving from **TClonesArray** to a normal object array will imply a very large performance penalty.

- Objects like **ATLFElectron, ATLFMuon, ATLFCluster,** etc. These classes derive from **TObject**. It is straightforward to remove the dependency on **TObject**. You loose the **"Inspect"** possibility.

- **ATLFDisplay** This class is totally based on the ROOT graphics system (2-D and 3-D).

- **All classes**

  In all **ATLFast++ classes**, basic data types are defined using the ROOT portable data types **Float_t, Int_t, Double_t,** etc instead of the C++ types float, int and double. **Double_t** for example guarantees always 64 bit precision, while double may be 64 or 128 bits. Changing these types is again straightforward. It will not require any changes at all in the source files, but only the addition of a small include file defining these types. Most classes use the **ROOT TMath class** for all mathematical functions (see the list we use below). **TMath** provides a very efficient implementation of the mathematical functions (sin, cos, sqrt, abs, sign, etc.). The computation is always in double precision, but the input parameters may be of different types. Most Maker classes use the ROOT **class TRandom** to get random numbers (**Rndm** in [0,1], or **Gaus**). These classes will have to be replaced by similar functions of another class library. The last dependency is the **ROOT histogramming** and **Tree package**. The **Tree facility** is unique to ROOT. It is based on the **TClonesArray implementation**.

- List of ROOT basic data types used in ATLfast++
  - Bool_t
  - Char_t
  - Short_t
  - Int_t
  - Float_t
  - Double_t

  For a complete list of all existing ROOT data types, see **http://root.cern.ch/root/html/ListOfTypes.html**

**The ROOT TClonessArray**

An array of clone (identical) objects. Memory for the objects stored in the array is allocated only once in the lifetime of the clones array. All objects must be of the same class. For the rest this class has the same properties as **TObjArray**.

This class is designed to reduce the very large number of new and delete calls in large loops like this (O(100000) x O(10000) times new/delete):

```
TObjArray a(10000);
while (TEvent *ev = (TEvent *)next()) {      // O(100000) events
 for (int i = 0; i < ev->Ntracks; i++) {   // O(10000) tracks
    a[i] = new TTrack(x,y,z,...);
    ...
 }
 ...
 a.Delete();
}
```

One better uses a TClonesArray which reduces the number of new/delete calls to only O(10000):

```
TCloneArray a("TTrack", 10000);
while (TEvent *ev = (TEvent *)next()) {      // O(100000) events
 for (int i = 0; i < ev->Ntracks; i++) {   // O(10000) tracks
    new(a[i]) TTrack(x,y,z,...);
    ...
    ...
 }
 ...
 a.Delete();
}
```

Considering that a new/delete costs about 70 mus, O(10^9) new/deletes will save about 19 hours.

- **The ATLFast HTML documentation**

  Root has a special class THtml to generate automatically an html version of your classes or macros. The ATLFast documentation on the Web at the **URL**

  **http:// root.cern.ch/root/html/atlfast/USER_Index.html**

  has been generated automatically in a Root interactive session with the following few commands:

  **Root > .x display.C**

  **Root > THtml html**

  **Root > html.MakeAll()**

- **Access to ATLFast classes via the ROOT interpreter**

  All **ATLFast** class header files are instrumented with a macro **ClassDef**. All Implementation files include a macro **ClassImp**. These two macros are expanded by the compiler to define a few additional member functions. The functions provide:

  - Run time access to any **ATLFast** class using the interpreter.
  - **Inspect** and **DrawClass** functionality.
  - Automatic **Input/Output**.
  - Automatic **html documentation**.

- **TPythia class**

  is part of the ROOT system. It is a C++ interface to the FORTRAN package PYTHIA. Running without ROOT will require a new class wrapper.