ROOT I/O Overview

CMS-ROOT meeting

CERN- October 10

René Brun

ftp://root.cern.ch/root/cms.ppt

http://root.cern.ch

Plan of talk

- Framework Structure
- The Object Dictionary and the rootcint tool
- Persistency basics:
  - Evolution of ROOT I/O
  - Basic Object I/O
  - CMS PSimHit example
  - File Structure
  - Streamers
  - Automatic Schema Evolution
- Special collection Classes
- Trees
  - Event example
  - PSimHit example
  - Tree Friends
  - Chains
- Folders
Motivations
Towards the ROOT Framework

Following our many years of experience with the development of the PAW system, we decided in 1995 to start the design and the implementation of a system capable of doing at least the same thing in an OO context, but also to serve as a complete framework from data taking to data analysis.

During a few months, we learnt the basics ingredients of an OO system by implementing several variants of an histograming package. We quickly implemented a rudimentary I/O sub system and also some very basic collection classes. It became rapidly clear to us that a more ambitious persistency mechanism had to be developed.

There was no point in developing a system supporting only the PAW CWNs in a world dealing with classes and complex object hierarchies. OODBMS could have been the solution to our problem, but we were convinced that the corresponding proposed commercial tools were not appropriate for a flexible data analysis environment.
Building a Modular System

Modularity is a buzzword with different meanings. A modular system is sometimes presented as a system with many small and independent components. In general, such systems do not have an object bus and the communication between the components is left to the application using these components.

Systems with a deep hierarchy of components may be difficult to maintain because of too many interdependencies between the top level and low level modules.

Is a system with well defined interfaces a modular system? Probably not, because too much emphasis is put on the interfaces at the expense of the object bus. In such systems, the interfaces may have long argument lists instead of well designed collections and object folders.

An end user will see a system as modular if the structure is easy to understand, while a system developer will put more emphasis on the maintenance aspects, probably the two aspects being strongly related. A modular system can also be seen as a system easy to integrate into another system.

After many iterations and user feedback, we have gradually converged to the following framework structure =>
ROOT Framework Organization

```
$ROOTSYS
bin          lib          tutorials          test          include

libCint.so   libCore.so   EditorBar.C    fitslicesy.C   ntuple1.C
libEG.so     *libEGPythia.so    formula1.C    oldbenchmarks.C
*libEGPythia6.so   *libEGVenus.so   framework.C    pdeg.dat
libGpad.so    libEG.so         games.C        pseexam.C
libGraf.so    *libEGGrafix.so    gaxis.C        psstable.C
libGraf3d.so  libGraf.so       geometry.C     rootalias.C
libGui.so     *libEGVenus.so    gerrors.C      rootenv.C
libGX11.so    *libGX11TTF.so    gerrors2.C     rootlogoff.C
libHist.so    libHistPainter.so    graph.C       rootlogon.C
libHtml.so    libHtml.so       h1draw.C       rootmarks.C
libMatrix.so  libMatrix.so     hadd.C         runcatalog.sql
libMinuit.so  libMinuit.so     hclient.C      rundemo.C
libNew.so     libNew.so        hcons.C        second.C
libPhysics.so libPhysics.so    hprod.C        shapes.C
libPostscript.so libPostscript.so    hserv.C       shared.C
libProofof.so libProofof.so    hserv2.C       splines.C
*libRFIO.so   *libRFIO.so       hsimple.C       sqcreatedb.C
*libRGL.so    *libRGL.so       hsum.C         sqfilldb.C
libRint.so    libRint.so       hsumTimer.C     sqselect.C
*libThread.so *libThread.so     htmlex.C       staff.C
libTree.so    libTree.so       io.C           staff.dat
libTreePlayer.so libTreePlayer.so    latex.C        surfaces.C
libTreeViewer.so libTreeViewer.so    latex2.C       tcl.C
*libttf.so    *libttf.so       latex3.C      测试random.C
libX3d.so     libX3d.so        manyaxis.C     tornado.C
libXpm.a      libXpm.a        multifit.C      tree.C

* Optional Installation

Aclock.cxx Aclock.h Event.cxx Event.h EventLinkDef.h Hello.cxx Hello.h
MainEvent.cxx Makefile Makefile.in Makefile.win32
README TestVectors.cxx Tetris.cxx Tetris.h
eventa.cxx eventb.cxx eventload.cxx guitest.cxx hsimple.cxx
hworld.cxx minexam.cxx stress.cxx
tcollbm.cxx tcollex.cxx
test2html.cxx tstring.cxx vlazy.cxx
vmatrix.cxx vvector.cxx
```
The Libraries

- Over 500 classes
- 650,000 lines of code
- Core (5 Mbytes)
- CINT (1.5 Mbytes)
- All libs (17 Mbytes)
- green libs linked on demand
Root Libs Structure

- Root libs are a layered structure
- CORE classes always required (support for RTTI, basic I/O and interpreter.
- The application libraries. You load only what you use. Separation between Data Objects and the high level classes acting on these objects. Example, a batch job uses only the Hist lib, no need to link HistPainter.
- Root shared libs reduce the application link time.
- Root libs are small libraries.
- Root libs can be used with other class libraries.
Dynamic Linking

A Shared Library can be linked dynamically to a running executable module

A Shared Library facilitates the development and maintenance phases

The "standard" ROOT executable module can dynamically load user's specific code from shared libraries.

```
Root > gSystem->Load("libNA49")
Root > gSystem->Load("libUser")
Root > T49Event event
Root > event.xxxxxxx
```

Dynamic linking from Shared libraries

**Application Executable Module**

**Experiment libraries**

**User libraries**

**General libraries**

**User libraries**
The ROOT Object Dictionary
The CINT RTTI

- The **Run Time Type Information** provided by CINT (rootcint) is the brain of Root. rootcint can be used to parse user classes and considerably extend the power of Root.

- RTTI is used by the I/O services

- By definition the interpreter is based on it.

- The GUI object context sensitive menus also.

- Also Browsers, Inspectors and html generator

- also Root utilities to draw class diagrams

- rootcint can be used to parse user classes such that user class functions can be called interactively and code for I/O generated automatically.
Rootcint Preprocessor

UserClass1.h

rootcint

UserCint.C

C++ code to create the RTTI

Interface for CINT interpreter

Streamers
Any User class library with the RTTI info can be plugged into a ROOT executable and its functions called interactively.

**Framework: Basic components**

<table>
<thead>
<tr>
<th>Lib1</th>
<th>RTTI: Objects Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input/Output</strong></td>
<td><strong>User Interface</strong></td>
</tr>
<tr>
<td>Stream full object</td>
<td><strong>Split mode</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Branches</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Attributes</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Event Loop</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Shared Memory</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Sockets</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Local data base</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Remote data base</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Gateways (Corba, RMI,..)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Keyboard, Mouse</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Sockets, Time</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Interpreter</strong></td>
</tr>
<tr>
<td></td>
<td><strong>GUI</strong></td>
</tr>
</tbody>
</table>

**High Level Components**

- 2-D/B-D graphics, Viewers, Browsers, Inspectors, Doc tools
- Histogramming, Minimisation, Ntuples, Trees
- Containers, Event iterators, Selectors

*idem for I/O*
Object Persistency
Simple to Complex cases

- Histograms
- Ntuples
- Trees
- Local Event Store
- Distributed Event Store
ROOT + RDBMS Model

- ROOT files
- Oracle MySQL
- Run/File Catalog
- Calibrations
- Event Store
  - histograms
  - Trees
  - Geometries
Ideal Persistency

- No constraints on object model
- Automatic schema evolution
- Remote access
- Automatic converters
- Efficient storage compression
- Granularity matching access patterns
- Machine independent format

transient

persistent

obj1;1, obj1;2, obj1;3
obj2;1, obj2;2
Evolution of ROOT I/O

- Hand-written Streamers
- Streamers generated via rootcint
- Support for Class Versions
- Support for ByteCount
- Several attempts to introduce automatic class evolution
- Persistent class Dictionary written to files
- rootcint modified to generate automatic Streamers
- can generate code for “DataObjects” classes in a file
- Support for STL and more complex C++ cases
- Trees take advantage of the new scheme
- Can read files without the classes
- Persistent Reference pointers (New)
ROOT I/O : An Example

Program Writing

TFile f("example.root","new");
TH1F h("h","My histogram",100,-3,3);
h.FillRandom("gaus",5000);
h.Write();

Program Reading

TFile f("example.root");
TH1F *h = (TH1F*)f.Get("h");
h->Draw();
f.Map();

20010831/171903 At:64 N=90 TFile
20010831/171941 At:154 N=453 TH1F CX = 2.09
20010831/171946 At:607 N=2364 StreamerInfo CX = 3.25
20010831/171946 At:2971 N=96 KeysList
20010831/171946 At:3067 N=56 FreeSegments
20010831/171946 At:3123 N=1 END
Example with CMS classes

- In the following slides we will use the CMS simulation classes PSimHit, etc.
- Classes suggested by Vincenzo as exercise.
- Goal: No changes in the class model
  - Minor changes in the header files to make these classes ROOT-aware
- Example1: How to generate the dictionary
- Example2: How to write PSimHit objects to a ROOT file (and read)
- Example3: How to write PSimHit objects to a Tree (and read)

ftp://root.cern.ch/root/cmsdemo.tar.gz
#include "LocalPoint.h"
#include "LocalVector.h"
#include "TObject.h"

class DetUnit;

class PSimHit : public TObject {
public:

PSimHit() : theDetUnitId(-1) {} 
PSimHit( const Local3DPoint& entry, const Local3DPoint& exit,
        float pabs, float tof, float eloss, int particleType,
        int detId, unsigned int trackId) :
    theEntryPoint( entry), theExitPoint(exit),
    ......

    float pabs() const {return thePabs;}
    float tof() const {return theTof;}
    float energyLoss() const {return theEnergyLoss;}
    int particleType() const {return theParticleType;}
    int detUnitId() const {return theDetUnitId;}
    unsigned int trackId() const {return theTrackId;}

private:

    // properties
    Local3DPoint theEntryPoint; // position
    Local3DPoint theExitPoint;
    float thePabs; // momentum
    float theTof; // Time Of Flight
    float theEnergyLoss; // Energy loss
    int theParticleType;

    // association
    int theDetUnitId;
    unsigned int theTrackId;

ClassDef(PSimHit,1)
};
#include "PV3DBase.h"
#include "Point2DBase.h"
#include "Vector3DBase.h"
#include "TObject.h"

template <class T, class FrameTag>
class Point3DBase : public PV3DBase< T, PointTag, FrameTag> { 
  public:
    typedef PV3DBase< T, PointTag, FrameTag> BaseClass;
    typedef Vector3DBase< T, FrameTag> VectorType;
    typedef Basic3DVector<T> BasicVectorType;

    Point3DBase() {}

    Point3DBase(const T& x, const T& y, const T& z) : BaseClass(x, y, z) {}

    ......
    ClassDefT(Point3DBase,1)
};
    ClassDef2T2(Point3DBase,T,FrameTag)
Point3DBase.h

```cpp
#include "Basic3DVector.h"
#include <iosfwd>
#include "TObject.h"

template <class T, class PVType, class FrameType>
class PV3DBase {
public:
    typedef Basic3DVector<T> BasicVectorType;

    PV3DBase() : theVector() {}
    PV3DBase(const T &x, const T &y, const T &z) : theVector(x, y, z) {}
    PV3DBase(const Basic3DVector<T>& v) : theVector(v) {}

    T x() const { return basicVector().x();}
    T y() const { return basicVector().y();}
    T mag2() const { return basicVector().mag2();}
    T r() const { return basicVector().r();}

    protected:
        BasicVectorType& basicVector() { return theVector;}

    private:
        BasicVectorType theVector;

    ClassDefT(PV3DBase,1)
};

ClassDef3T2(PV3DBase,T,PVType,FrameType)
```
```cpp
#include "Basic2DVector.h"
#include <iosfwd>
#include <cmath>
#include "TObject.h"

template < class T>
class Basic3DVector { 

public: 
    // default constructor
    Basic3DVector() : theX(0), theY(0), theZ(0){} 

    Basic3DVector( const T& x, const T& y, const T& z) :
        theX(x), theY(y), theZ(z) {}  

    T x() const { return theX;}
    T y() const { return theY;}
    T z() const { return theZ;}

    ...

private:
    T theX;
    T theY;
    T theZ;

ClassDefT(Basic3DVector,1)
};
ClassDefT2(Basic3DVector,T)
```
Running rootcint on PSimHit.h
Building the shared lib

```bash
rootcint -f Dict.cxx -c PSimHit.h LinkDef.h

g++ -fPIC -I$ROOTSYS/include -c Dict.cxx

g++ -fPIC -I$ROOTSYS/include -c PSimHit.cxx

g++ -shared -g PSimHit.o Dict.o -o libHit.so
```

```
#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;

#pragma link C++ class LocalTag+;
#pragma link C++ class PointTag+;
#pragma link C++ class VectorTag+;
#pragma link C++ class Basic2DVector<float>+;
#pragma link C++ class Basic3DVector<float>+;
#pragma link C++ class PV2DBase<float, VectorTag, LocalTag>+;
#pragma link C++ class PV2DBase<float, PointTag, LocalTag>+;
#pragma link C++ class PV3DBase<float, VectorTag, LocalTag>+;
#pragma link C++ class PV3DBase<float, PointTag, LocalTag>+;
#pragma link C++ class Vector2DBase<float, LocalTag>+;
#pragma link C++ class Vector3DBase<float, LocalTag>+;
#pragma link C++ class Point2DBase<float, LocalTag>+;
#pragma link C++ class Point3DBase<float, LocalTag>+;
#pragma link C++ class PSimHit+;
```

LinkDef.h

Classes used by PSimHit
use C++ templates
heavily
All Template instances
must be declared
void demo1() {
    //create a new ROOT file
    TFile f("demo1.root","recreate");

    //Create a PSimHit with the default constructor
    PSimHit h1;

    //Write it to the file with the key name hit1
    h1.Write("hit1");

    //Create a normal PSimHit with the entry and exit point
    Local3DPoint pentry(1,2,3);
    Local3DPoint pexit(10,20,30);
    float pabs = 41;
    float tof = 1.67e-8;
    float eloss = 5.78e-3;
    int pType = 12;
    int detId = 67;
    int trackId = 1234;
    PSimHit h2(pentry,pexit,pabs,tof,eloss,pType,detId,trackId);

    //Write it to the file with the key name hit2
    h2.Write("hit2");
}

Writing CMS PSimHit objects
void demo2() {
  //connect the ROOT file demo1.root in readonly mode
  TFile *f = new TFile("demo1.root");

  //Read hit2
  PSimHit *hit = (PSimHit*)f->Get("hit2");

  //print some hit members
  cout <<" X1= " <<hit->entryPoint().x() <<" Y2= " <<hit->exitPoint().y() <<" pabs= " <<hit.pabs() <<endl;
  delete hit;

  //Open the ROOT browser and inspect the file
  new TBrowser;

  //click on "ROOT files", then "demo1.root", with the right button, select menu items "Inspect", "DrawClass"
  //on hit2
}

Browsing the file

```c
root [0] TFile f("demo1.root")
root [1] TBrowser b
```

The description of all classes in a file is written in one single record when the file is closed:

StreamerInfo
ROOT I/O -- Sequential/Flat

transient object is serialized by the streamer

No need for transient/persistent classes

TFile

TMapFile
shared memory

TRFIOFile
RFIO daemon

TBuffer

TWebFile
web server

TNtFile
rootd

http

sockets
All what you need to known to navigate in a ROOT file
Root objects or any User Object can be stored in ROOT folders and browsed.
ROOT files can be structured like a Unix file system.
A Root file `pippa.root` with two levels of directories.

Objects in directory `/pippa/DM/CJ` eg: `/pippa/DM/CJ/h15`
LAN/WAN files

- **Files and Directories**
  - a directory holds a list of named objects
  - a file may have a hierarchy of directories (a la Unix)
  - ROOT files are machine independent
  - built-in compression

- **Support for local, LAN and WAN files**
  - TFile f1("myfile.root")
  - TFile f2("http://pcbrun.cern.ch/Renefile.root")
  - TFile f3("root://cdfsga.fnal.gov/bigfile.root")
  - TFile f4("rfio://alice/run678.root")

See Fons talk
Streaming Objects
Old Streamers in 0.90 (1996)
Evolution illustrated with the ROOT class TAxis

class TAxis : public TNamed, public TAttAxis {

private:
  Int_t fNbins;
  Float_t fXmin;
  Float_t fXmax;
  TArrayF fXbins;

void TAxis::Streamer(TBuffer &b)
{
  if (b.IsReading()) {
    Version_t v = b.ReadVersion();
    TNamed::Streamer(b);
    TAttAxis::Streamer(b);
    b >> fNbins;
    b >> fXmin;
    b >> fXmax;
    fXbinsStreamer(b);
  } else {
    b.WriteVersion(TAxis::IsA());
    TNamed::Streamer(b);
    TAttAxis::Streamer(b);
    b << fNbins;
    b << fXmin;
    b << fXmax;
    fXbinsStreamer(b);
  }
}

TBuffer b;
object.Streamer(b);
Old Streamers in 2.25 (1999)

```cpp
class TAxis : public TNamed, 
            public TAttAxis {

    private:
        Int_t     fNbins;
        Float_t   fXmin;
        Float_t   fXmax;
        TArrayF   fXbins;
        Int_t     fFirst;
        Int_t     fLast;
        TString   fTimeFormat;
        Bool_t    fTimeDisplay;

    void TAxis::Streamer(TBuffer &b) {
        UInt_t R__s, R__c;
        if (b.IsReading()) {
            Version_t v = b.ReadVersion(&R__s, &R__c);
            TNamed::Streamer(b);
            TAttAxis::Streamer(b);
            b >> fNbins;
            b >> fXmin;
            b >> fXmax;
            fXbinsStreamer(b);
            if (v > 2) {
                b >> fFirst;
                b >> fLast;
            }
            if (v > 3) {
                b >> fTimeDisplay;
                fTimeFormatStreamer(b);
            } else {
                SetTimeFormat();
                b.WriteByteCount(R__s, R__c, TAxis::IsA());
            } else {
                R__c = b.WriteVersion(TAxis::IsA(), kTRUE);
                TNamed::Streamer(b);
                TAttAxis::Streamer(b);
                b <<= fNbins;
                b <<= fXmin;
                b <<= fXmax;
                fXbinsStreamer(b);
                b <<= fFirst;
                b <<= fLast;
                b <<= fTimeDisplay;
                fTimeFormatStreamer(b);
                b.SetByteCount(R__c, kTRUE);
            }
        }
    }
}
```
Problems with Old Streamers

- Experience in several large experiments has shown that a system based only on automatic code generation with no support for schema evolution is not a long term solution. A huge maintenance problem.

- In a system with several hundred (thousand) classes and as many users, it is difficult to maintain coherent shared libs to support all possible combinations when accessing collections of old data sets.

- A few attempts (eg in STAR) to support automatic schema evolution seen as a progress, but not sufficient.

- We have seen a rapidly growing request for reading data sets without having the original classes.

- **Backward compatibility** (reading an old data set with new classes) is a must. **Forward compatibility** (reading a new data set with old classes) also a must.
The ROOT solution

- Minimize reliance on generated code.
- Exploit the powerful CINT Object Dictionary
- Make the process as automatic as possible and as simple as possible.
- Be as efficient as with the generated code.
- Self-describing data sets.
- Come with a solution that does not prevent the move to another language in the future.
- Back compatibility with the original system.
  - Like upgrading the engine in a running car

Implementing all these features was a non trivial exercise and a lot of work

Thanks to our huge users base for providing many use cases and testing
New Streamers in 3.00

class TAxis : public TNamed,  
    public TAttAxis {

private:
    Int_t   fNbins;
    Double_t fXmin;
    Double_t fXmax;
    TArrayD fXbins;
    Char_t *fXlabels;  ///<
    Int_t   fFirst;
    Int_t   fLast;
    TString fTimeFormat;
    Bool_t  fTimeDisplay;
    TObject *fParent;  ///

    void TAxis::Streamer(TBuffer &b) {
        // Stream an object of class TAxis.

        if (b.IsReading())
            TAxis::Class()->ReadBuffer(b, this);
        else
            TAxis::Class()->WriteBuffer(b, this);
    }
ootcint
enum {kSize=10};

char fType[20];  // array of 20 chars
Int_t fNtrack;   // number of tracks
Int_t fNvertex;  // number of vertices
Int_t fX[kSize]; // an array where dimension is an enum
UInt_t fFlag;    // bit pattern event flag
Float_t fMatrix[4][4]; // a two-dim array
Float_t *fDistance;  // [fNvertex] array of floats of length fNvertex
Double_t fTemperature; // event temperature
TString *fTstringp; // [fNvertex] array of TString
TString fNames[12]; // array of TString
TAxis fXaxis; // example of class derived from TObject
TAxis fYaxis[3]; // array of objects
TAxis *fVaxis[3]; // pointer to an array of TAxis
TAxis *fPaxis;   // [fNvertex] array of TAxis of length fNvertex
TAxis **fQaxis;  // [fNvertex] array of pointers to TAxis objects
TDatime fDatime; // date and time
EventHeader fEvtHdr; // example of class not derived from TObject
TObjArray fObjArray; // An object array of TObject*
TClonesArray *fTracks; // -- > array of tracks
TH1F *fH;             // -- > pointer to an histogram
TArrayF fArrayF;      // an array of floats
TArrayI *fArrayI;     // a pointer to an array of integers

...................(see next)
## Support for STL

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector&lt;int&gt;</td>
<td>fVectorint;</td>
<td>STL vector on ints</td>
</tr>
<tr>
<td>vector&lt;short&gt;</td>
<td>fVectorshort;</td>
<td>STL vector of shorts</td>
</tr>
<tr>
<td>vector&lt;double&gt;</td>
<td>fVectorD[4];</td>
<td>array of STL vectors of doubles</td>
</tr>
<tr>
<td>vector&lt;TLine&gt;</td>
<td>fVectorTLine;</td>
<td>STL vector of TLine objects</td>
</tr>
<tr>
<td>vector&lt;TObject&gt;</td>
<td>*fVectorTObject;</td>
<td>pointer to an STL vector</td>
</tr>
<tr>
<td>vector&lt;TNamed&gt;</td>
<td>*fVectorTNamed[6];</td>
<td>array of pointers to STL vectors</td>
</tr>
<tr>
<td>deque&lt;TAttLine&gt;</td>
<td>fDeque;</td>
<td>STL deque</td>
</tr>
<tr>
<td>list&lt;const TObject*&gt;</td>
<td>fVectorTObjectp;</td>
<td>STL list of pointers to objects</td>
</tr>
<tr>
<td>list&lt;string&gt;</td>
<td>*fListString;</td>
<td>STL list of strings</td>
</tr>
<tr>
<td>list&lt;string *&gt;</td>
<td>fListStringp;</td>
<td>STL list of pointers to strings</td>
</tr>
<tr>
<td>map&lt;TNamed*,int&gt;</td>
<td>fMapTNamedp;</td>
<td>STL map</td>
</tr>
<tr>
<td>map&lt;TString,TList*&gt;</td>
<td>fMapList;</td>
<td>STL map</td>
</tr>
<tr>
<td>map&lt;TAxis*,int&gt;</td>
<td>*fMapTAxisp;</td>
<td>pointer to STL map</td>
</tr>
<tr>
<td>set&lt;TAxis*&gt;</td>
<td>fSetTAxis;</td>
<td>STL set</td>
</tr>
<tr>
<td>set&lt;TAxis*&gt;</td>
<td>*fSetTAxisp;</td>
<td>pointer to STL set</td>
</tr>
<tr>
<td>multimap&lt;TNamed*,int&gt;</td>
<td>fMultiMapTNamedp;</td>
<td>STL multimap</td>
</tr>
<tr>
<td>multiset&lt;TAxis*&gt;</td>
<td>*fMultiSetTAxisp;</td>
<td>pointer to STL multiset</td>
</tr>
<tr>
<td>string</td>
<td>fString;</td>
<td>C++ standard string</td>
</tr>
<tr>
<td>string</td>
<td>*fStringp;</td>
<td>pointer to standard C++ string</td>
</tr>
<tr>
<td>UShortVector</td>
<td>fUshort;</td>
<td>class with an STL vector as base class</td>
</tr>
</tbody>
</table>
Complex STL use not supported

- `vector<vector<TAxis *> > fVectAxis;` //STL vector of vectors of TAxis*
- `map<string,vector<int> > fMapString;` //STL map of string/vector
- `deque<pair<float,float> > fDequePair;` //STL deque of pair

Use a custom Streamer for these complex cases
All ROOT collections support **Polymorphism**

- **TCollection** (abstract base class)
- **TSeqCollection, TList, THashList**
- **TMap, TExMap**
- **TObjArray**

**TClonesArray** is a specialized collection for arrays of objects of the same class.

It minimizes the overhead due to new/delete. Much more efficient than STL for I/O (see next slides)

**TRefArray** is an optimized collection for persistent reference pointers.

See example Event

```c
Tbuffer b;
collectionStreamer(b);
```
The Test suite “bench”
(example on fcdfs gui2 with KAI compiler)

- Test performance of STL vector of objects, vectors of pointers and same with a TClonesArray of TObjHit deriving from THit

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>cx</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector&lt;THit&gt;</td>
<td>2.16</td>
<td>1.91</td>
<td>4.57</td>
</tr>
<tr>
<td>vector&lt;THit*&gt;</td>
<td>2.36</td>
<td>1.86</td>
<td>4.56</td>
</tr>
<tr>
<td>TClonesArray(TObjHit)</td>
<td>1.98</td>
<td>1.62</td>
<td>6.77</td>
</tr>
<tr>
<td>TClonesArray(TObjHit) split</td>
<td>1.98</td>
<td>1.62</td>
<td>6.76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size of file in bytes</th>
<th>comp 0</th>
<th>Reference</th>
<th>comp 1</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector&lt;THit&gt;</td>
<td>42053031</td>
<td>42053031</td>
<td>9213642</td>
<td>9213459</td>
</tr>
<tr>
<td>vector&lt;THit*&gt;</td>
<td>42079941</td>
<td>42079941</td>
<td>9220556</td>
<td>9215935</td>
</tr>
<tr>
<td>TClonesArray(TObjHit)</td>
<td>39807325</td>
<td>39807325</td>
<td>58978130</td>
<td>5892837</td>
</tr>
<tr>
<td>TClonesArray(TObjHit) split</td>
<td>39807325</td>
<td>39807325</td>
<td>5890726</td>
<td>5901163</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time to write in seconds</th>
<th>comp 0</th>
<th>Reference</th>
<th>comp 1</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector&lt;THit&gt;</td>
<td>2.63</td>
<td>1.74</td>
<td>9.34</td>
<td>9.58</td>
</tr>
<tr>
<td>vector&lt;THit*&gt;</td>
<td>2.47</td>
<td>1.80</td>
<td>9.44</td>
<td>9.62</td>
</tr>
<tr>
<td>TClonesArray(TObjHit)</td>
<td>1.33</td>
<td>1.60</td>
<td>5.45</td>
<td>7.32</td>
</tr>
<tr>
<td>TClonesArray(TObjHit) split</td>
<td>1.23</td>
<td>1.51</td>
<td>5.46</td>
<td>6.18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time to read in seconds</th>
<th>comp 0</th>
<th>Reference</th>
<th>comp 1</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector&lt;THit&gt;</td>
<td>3.03</td>
<td>2.29</td>
<td>4.24</td>
<td>3.67</td>
</tr>
<tr>
<td>vector&lt;THit*&gt;</td>
<td>3.01</td>
<td>2.10</td>
<td>4.28</td>
<td>3.27</td>
</tr>
<tr>
<td>TClonesArray(TObjHit)</td>
<td>1.34</td>
<td>1.53</td>
<td>1.88</td>
<td>2.14</td>
</tr>
<tr>
<td>TClonesArray(TObjHit) split</td>
<td>1.35</td>
<td>1.35</td>
<td>1.88</td>
<td>1.94</td>
</tr>
</tbody>
</table>

| Total CPU time          | 82.14  | 76.33     |        |           |
| Estimated ROOTMARKS     | 185.85 | 200.00    |        |           |

Better compression with TClonesArray
Better write with TClonesArray
Much better read with TClonesArray
Self-Describing file

Root > TFile f("demo1.root");
Root > f.ShowStreamerInfo();

StreamerInfo for class: PSimHit, version=1
  BASE TObject offset= 0 type=66 Basic ROOT object
  Local3DPoint theEntryPoint offset= 0 type=62 position
  Local3DPoint theExitPoint offset= 0 type=62
  float thePabs offset= 0 type=5 momentum
  float theTof offset= 0 type=5 Time Of Flight
  float theEnergyLoss offset= 0 type=5 Energy loss
  int theParticleType offset= 0 type=3
  int theDetUnitId offset= 0 type=3
  unsigned int theTrackId offset= 0 type=13

StreamerInfo for class: Point3DBase<float,LocalTag>, version=1
  BASE PV3DBase<float,PointTag,LocalTag> offset= 0 type=0

StreamerInfo for class: PV3DBase<float,PointTag,LocalTag>, version=1
  Basic3DVector<float> theVector offset= 0 type=62

StreamerInfo for class: Basic3DVector<float>, version=1
  float theX offset= 0 type=5
  float theY offset= 0 type=5
  float theZ offset= 0 type=5
Self-describing files

- Dictionary for persistent classes written to the file when closing the file.
- ROOT files can be read by foreign readers (e.g., JavaRoot (Tony Johnson))
- Support for Backward and Forward compatibility
- Files created in 2003 must be readable in 2015
- Classes (data objects) for all objects in a file can be regenerated via `TFile::MakeProject`

```c
Root > TFile f("demo.root");
Root > f.MakeProject("dir","*","new++");
```
Showing classes in a file

TFile::ShowStreamerInfo()

StreamInfo for class: ATLFMuon, version=1

| BASE TObject offset= 0 type=66 Basic ROOT object |
| BASE TAtt3D offset= 0 type=0 3D attributes |
| int_t m_KFcode offset= 0 type=3 Muon KF-code |
| int_t m_MCParticle offset= 0 type=3 Muon position in MCParticles list |
| int_t m_MotherKF offset= 0 type=3 Muon mother KF-code |
| int_t m_UserFlag offset= 0 type=3 Muon energy usage flag (0 for used in clusters) |
| int_t m_Isolated offset= 0 type=3 Muon isolation (1 for isolated) |
| float_t m_Eta offset= 0 type=5 Eta coordinate |
| float_t m_Phi offset= 0 type=5 Phi coordinate |
| float_t m_PT offset= 0 type=5 Transverse energy |
| int_t m_Trigger offset= 0 type=3 Result of trigger |

StreamInfo for class: ATLFElectron, version=1

| BASE TObject offset= 0 type=66 Basic ROOT object |
| BASE TAtt3D offset= 0 type=0 3D attributes |
| int_t m_KFcode offset= 0 type=3 Electron KF-code |
| int_t m_MCParticle offset= 0 type=3 Electron position in MCParticles list |
| int_t m_MotherKF offset= 0 type=3 Electron mother KF-code |
| float_t m_Eta offset= 0 type=5 Eta coordinate |
| float_t m_Phi offset= 0 type=5 Phi coordinate |
| float_t m_PT offset= 0 type=5 Transverse energy |

StreamInfo for class: ATLP Photon, version=1

| BASE TObject offset= 0 type=66 Basic ROOT object |
| BASE TAtt3D offset= 0 type=0 3D attributes |
| int_t m_KFcode offset= 0 type=3 Photon KF-code |
| int_t m_MCParticle offset= 0 type=3 Photon position in MCParticles list |
| int_t m_MotherKF offset= 0 type=3 Photon mother KF-code |
| float_t m_Eta offset= 0 type=5 Eta coordinate |
| float_t m_Phi offset= 0 type=5 Phi coordinate |
| float_t m_PT offset= 0 type=5 Transverse energy |

StreamInfo for class: ATLP Jet, version=1

| BASE TObject offset= 0 type=66 Basic ROOT object |
| BASE TAtt3D offset= 0 type=0 3D attributes |
| int_t m_KFcode offset= 0 type=3 Jet KF-code |
| int_t m_NCells offset= 0 type=3 Number of cells used for reconstruction |
| int_t m_NParticles offset= 0 type=3 Number of particles assigned to jet |
| int_t m_Position offset= 0 type=3 Position in MCParticle list of matching b-quark/c quark |
| float_t m_Eta0 offset= 0 type=5 Eta position of initiator cell |
| float_t m_Phi0 offset= 0 type=5 Phi position of initiator cell |
| float_t m_Eta offset= 0 type=5 Eta of jet bary-center |
| float_t m_Phi offset= 0 type=5 Phi of jet bary-center |
| float_t m_PT offset= 0 type=5 Transverse momentum of jet |

....
Automatic Schema Evolution

1) An old version of a shared library and a file with new class definitions. This can be the case when someone has not updated the library and is reading a new file.

2) Reading a file with a shared library that is missing a class definition (i.e., missing class D).

3) Reading a file without any class definitions. This can be the case where the class definition is lost, or unavailable.

4) The current version of a shared library and an old file with old class versions (backward compatibility). This is often the case when reading old data.

5) Reading a file with a shared library built with MakeProject. This is the case when someone has already read the data without a shared library and has used ROOT’s MakeProject feature to reconstruct the class definitions and shared library (MakeProject is explained in detail later on).
Auto Schema Evolution (2)

In case of a mismatch between the in-memory version and the persistent version of a class, ROOT maps the persistent one to the one in memory. This allows you to change the class definition at will, for example:

1) Change the order of data members in the class.
2) Add new data members. By default the value of the missing member will be 0 or in case of an object it will be set to null.
3) Remove data members.
4) Move a data member to a base class or vice versa.
5) Change the type of a member if it is a simple type or a pointer to a simple type. If a loss of precision occurs, a warning is given.
6) Add or remove a base class
ROOT Trees
**Ntuples and Trees**

- **Ntuples**
  - support PAW-like ntuples and functions
  - PAW ntuples/histograms can be imported

- **Trees**
  - Extension of Ntuples for Objects
  - Collection of branches (branch has its own buffer)
  - Can input partial Event
  - Can have several Trees in parallel

- **Chains** = collections of Trees
Why Trees?

- Any object deriving from TObject can be written to a file with an associated key with `object.Write()`.
- However, each key has an overhead in the directory structure in memory (about 60 bytes). `Object.Write` is very convenient for objects like histograms, detector objects, calibrations, but not for event objects.
Why Trees?

- Trees have been designed to support very large collections of objects. The overhead in memory is in general less than 4 bytes per entry.
- Trees allow direct and random access to any entry (sequential access is the best)
- Trees have branches and leaves. One can read a subset of all branches. This can speed-up considerably the data analysis processes.
Why Trees?

- PAW ntuples are a special case of Trees.
- Trees are designed to work with complex event objects.
- High level functions like `TTree::Draw` loop on all entries with selection expressions.
- Trees can be browsed via `TBrowser`
- Trees can be analyzed via `TTreeViewer`

The PROOF system is designed to process chains of Trees in parallel in a GRID environment.
Create a TTree Object

A tree is a list of branches.

The TTree Constructor:

- Tree Name (e.g. "myTree")
- Tree Title

```
TTree *tree = new TTree("T","A ROOT tree");
```
**Adding a Branch**

- Branch name
- Class name
- Address of the pointer to the Object (descendant of TObject)
- Buffer size (default = 32,000)
- Split level (default = 1)

```cpp
Event *event = new Event();
myTree->Branch("eBranch","Event", &event, 64000, 1);
```
Splitting a Branch

Setting the split level (default = 1)

Example:

```c++
tree->Branch("EvBr", "Event", &ev, 64000, 0);
```
Adding Branches with a List of Variables

- Branch name
- Address: the address of the first item of a structure.
- Leaflist: all variable names and types
- Order the variables according to their size

Example

```c
TBranch *b = tree->Branch("Ev_Branch", &event, "ntrack/I:nseg:nvtex:flag/i:temp/F");
```
Adding Branches with a TClnonesArray

- Branch name
- Address of a pointer to a TClnonesArray
- Buffer size
- Split level (default = 1)

Example:
```c++
tree->Branch( "Track_B", &Track, 64000, 1);
```
Filling the Tree

- Create a for loop
- Create Event objects.
- Call the Fill method for the tree.
  
  `myTree->Fill()`
Write the Tree header

The Tree header contains a description of the Tree

- It owns the collection of branches
- Each branch has a buffer (TBasket) partially filled
- TTree::Write writes one single record on the file

```cpp
  tree->Write();
```
ROOT I/O -- Split/Cluster

Clustering per attribute or sub-object

Object in memory

Streamer

Object in memory

Branches

File
In Split mode, objects of the same type are automatically sorted. This makes selective reading much faster.
ROOT I/O - Split - multifile

Streamer

Object in memory

Object in memory

File1

File2

File3

TAGs

Tapes
Same Object Model + Choice of Buffering techniques

Serial I/O : TBuffer

Serial mode

TPC Tracks TClonesArray

Split mode

SPLIT mode: Buffers are Branch baskets

/user/brun/root/slides/chep97/iomode.C
Structure designed to support very large DBs
The Event class

class Event : public TObject {

private:
    char fType[20]; //event type
    Int_t fNtrack; //Number of tracks
    Int_t fNseg; //Number of track segments
    Int_t fNvertex;
    UInt_t fFlag;
    Float_t fTemperature;
    Int_t fMeasures[10];
    Float_t fMatrix[4][4];
    Float_t *fClosestDistance; //fNvertex
    EventHeader fEvtHdr;
    TClonesArray *fTracks; //->array with all tracks
    TRefArray *fHighPt; //array of High Pt tracks only
    TRefArray *fMuons; //array of Muon tracks only
    TRef fLastTrack; //reference pointer to last track
    TH1F *fH; //->

See $ROOTSYS/test/Event.h
The Track class

class Track : public TObject {

private:
    Float_t fPx; //X component of the momentum
    Float_t fPy; //Y component of the momentum
    Float_t fPz; //Z component of the momentum
    Float_t fRandom; //A random track quantity
    Float_t fMass2; //The mass square of this particle
    Float_t fBx; //X intercept at the vertex
    Float_t fBy; //Y intercept at the vertex
    Float_t fMeanCharge; //Mean charge deposition of all hits
    Float_t fXfirst; //X coordinate of the first point
    Float_t fXlast; //X coordinate of the last point
    Float_t fYfirst; //Y coordinate of the first point
    Float_t fYlast; //Y coordinate of the last point
    Float_t fZfirst; //Z coordinate of the first point
    Float_t fZlast; //Z coordinate of the last point
    Float_t fCharge; //Charge of this track
    Float_t fVertex[3]; //Track vertex position
    Int_t fNpoint; //Number of points for this track
    Short_t fValid; //Validity criterion
void Event::Build(Int_t ev, Int_t ntrack, Float_t ptmin) {
    Clear();
    for (Int_t t = 0; t < ntrack; t++) AddTrack(random, ptmin);
}

Track *Event::AddTrack(Float_t random, Float_t ptmin)
{
    // Add a new track to the list of tracks for this event.
    // To avoid calling the very time consuming operator new for each track,
    // the standard but not well know C++ operator "new with placement"
    // is called. If tracks[i] is 0, a new Track object will be created
    // otherwise the previous Track[i] will be overwritten.

    TClonesArray &tracks = *fTracks;
    Track *track = new(tracks[fNtrack++]) Track(random);  
    //Save reference to last Track in the collection of Tracks
    fLastTrack = track;
    //Save reference in fHighPt if track is a high Pt track
    if (track->GetPt() > ptmin) fHighPt->Add(track);
    //Save reference in fMuons if track is a muon candidate
    if (track->GetMass2() < 0.11) fMuons->Add(track);
    return track;
}
void demoe(int nevents) {
    //load shared lib with the Event class
    gSystem->Load("$ROOTSYS/test/libEvent");

    //create a new ROOT file
    TFile f("demoe.root","new");

    //Create a ROOT Tree with one single top level branch
    int split = 99;  //try also split=1 and split=0
    int bufsize = 16000;
    Event *event = new Event;
    TTree T("T","Event demo tree");
    T.Branch("event","Event",&event,bufsize,split);

    //Build Event in a loop and fill the Tree
    for (int i=0;i<nevents;i++) {
        event->Build(i);
        T.Fill();
    }

    T.Print();  //Print Tree statistics
    T.Write();  //Write Tree header to the file
}

All the examples can be executed with CINT or the compiler
root > .x demoe.C
root > .x demoe.C++
void demoer() {
    //load shared lib with the Event class
    gSystem->Load("$ROOTSYS/test/libEvent");

    //connect ROOT file
    TFile *f = new TFile("demoe.root");

    //Read Tree header and set top branch address
    Event *event = 0;
    TTree *T = (TTree*)f->Get("T");
    T->SetBranchAddress("event", &event);

    //Loop on events and fill an histogram
    TH1F *h = new TH1F("hntrack","Number of tracks",100,580,620);
    int nevents = (int)T->GetEntries();
    for (int i=0; i<nevents; i++) {
        T->GetEntry(i);
        h->Fill(event->GetNtrack());
    }

    h->Draw();
}
void demoer2() {
    //load shared lib with the Event class
    gSystem->Load("$ROOTSYS/test/libEvent");

    //connect ROOT file
    TFile *f = new TFile("demoe.root");

    //Read Tree header and set top branch address
    Event *event = 0;
    TTree *T = (TTree*)f->Get("T");
    T->SetBranchAddress("event", &event);
    TBranch *bntrack = T->GetBranch("fNtrack");

    //Loop on events and fill an histogram
    TH1F *h = new TH1F("hntrack","Number of tracks",100,580,620);
    int nevents = (int)T->GetEntries();
    for (int i=0;i<nevents;i++) {
        bntrack->GetEntry(i);
        h->Fill(event->GetNtrack());
    }

    h->Draw();
}
void demoer3() {
    //load shared lib with the Event class
    gSystem->Load("$ROOTSYS/test/libEvent");

    //connect ROOT file
    TFile *f = new TFile("demoe.root");

    //Read Tree header
    TTree *T = (TTree*)f->Get("T");

    //Histogram number of tracks via the TreePlayer
    T->Draw("event->GetNtrack()");
}
Writing CMS PSimHit in a Tree

```c
void demo3() {
    //create a new ROOT file
    TFile f("demo3.root","recreate");

    //Create a ROOT Tree with one single top level branch
    int split = 99;  //you can try split=1 and split=0
    int bufsize = 16000;
    PSimHit *hit = 0;
    TTree T("T","CMS demo tree");
    T.Branch("hit","PSimHit",&hit,bufsize,split);

    //Create hits in a loop and fill the Tree
    TRandom r;
    for (int i=0;i<50000;i++) {
        delete hit;
        Local3DPoint pentry(r.Gaus(0,1), r.Gaus(0,1), r.Gaus(0,10));
        Local3DPoint pexit (r.Gaus(0,3), r.Gaus(0,3), r.Gaus(50,20));
        float pabs = 100*r.Rndm();
        float tof = r.Gaus(1e-6,1e-8);
        float eloss= r.Landau(1e-3,1e-7);
        int ptype = i%2;
        int detId = i%20;
        int trackId= i%100;
        hit = new PSimHit(pentry,pexit,pabs,tof,eloss,ptype,detId,trackId);

        T.Fill();
    }

    T.Print();  //Print Tree statistics
    T.Write();   //Write Tree header to the file
}
```
Browsing the PSimHit Tree

split = 0

**Tree** : T 
**Entries** : 50000 

**Entries** : hit 
**Baskets** : 295 

1 branch only
Browsing the PSimHit Tree

split = 1

9 branches
### Browsing the PSimHit Tree

**split = 99**

#### Tree: CMS demo tree
- **Entries**: 50000
- **Total Size**: 268752 bytes
- **File Size**: 1509041 bytes
- **Tree compression factor**: 1.78

#### Branch: hit
- **Entries**: 50000
- **BranchElement (see below)**

<table>
<thead>
<tr>
<th>Branch</th>
<th>Description</th>
<th>Entries</th>
<th>Total Size</th>
<th>File Size</th>
<th>Baskets</th>
<th>Basket Size</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Br 0</td>
<td>fUniqueID</td>
<td>50000</td>
<td>191964 bytes</td>
<td>1276</td>
<td>12</td>
<td>16000 bytes</td>
<td>152.92</td>
</tr>
<tr>
<td>Br 1</td>
<td>fBits</td>
<td>50000</td>
<td>191964 bytes</td>
<td>1260</td>
<td>12</td>
<td>16000 bytes</td>
<td>152.35</td>
</tr>
<tr>
<td>Br 2</td>
<td>theEntryPoint</td>
<td>50000</td>
<td>191952 bytes</td>
<td>177959</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.00</td>
</tr>
<tr>
<td>Br 3</td>
<td>theEntryPoint.vector.thX</td>
<td>50000</td>
<td>191952 bytes</td>
<td>177934</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.00</td>
</tr>
<tr>
<td>Br 4</td>
<td>theEntryPoint.vector.thZ</td>
<td>50000</td>
<td>191988 bytes</td>
<td>178312</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.00</td>
</tr>
<tr>
<td>Br 5</td>
<td>theEntryPoint.vector.thY</td>
<td>50000</td>
<td>191952 bytes</td>
<td>178072</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.00</td>
</tr>
<tr>
<td>Br 6</td>
<td>theEntryPoint.vector.thE</td>
<td>50000</td>
<td>191952 bytes</td>
<td>164855</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.14</td>
</tr>
<tr>
<td>Br 7</td>
<td>theEntryPoint.vector.thZ</td>
<td>50000</td>
<td>191952 bytes</td>
<td>170871</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.00</td>
</tr>
<tr>
<td>Br 8</td>
<td>theEntryPoint.vector.thY</td>
<td>50000</td>
<td>191952 bytes</td>
<td>170771</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.00</td>
</tr>
<tr>
<td>Br 9</td>
<td>theEntryPoint.vector.thE</td>
<td>50000</td>
<td>191952 bytes</td>
<td>164544</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.12</td>
</tr>
<tr>
<td>Br 10</td>
<td>theEntryPoint.vector.thX</td>
<td>50000</td>
<td>191952 bytes</td>
<td>163207</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.12</td>
</tr>
<tr>
<td>Br 11</td>
<td>theTrackId</td>
<td>50000</td>
<td>191976 bytes</td>
<td>145544</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.32</td>
</tr>
<tr>
<td>Br 12</td>
<td>theEnergyLoss</td>
<td>50000</td>
<td>191964 bytes</td>
<td>122765</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.56</td>
</tr>
<tr>
<td>Br 13</td>
<td>theParticleType</td>
<td>50000</td>
<td>191964 bytes</td>
<td>122765</td>
<td>12</td>
<td>16000 bytes</td>
<td>1.56</td>
</tr>
<tr>
<td>Br 14</td>
<td>theBeamId</td>
<td>50000</td>
<td>191952 bytes</td>
<td>2294</td>
<td>12</td>
<td>16000 bytes</td>
<td>83.53</td>
</tr>
<tr>
<td>Br 15</td>
<td>theTrackId</td>
<td>50000</td>
<td>191976 bytes</td>
<td>4177</td>
<td>12</td>
<td>16000 bytes</td>
<td>45.94</td>
</tr>
</tbody>
</table>

---

**Double click produces this histogram**

**16 branches**
Collections of Hits

- A more realistic Tree will have
  - A collection of Detectors
  - Each detector one or more collection of hits
36 branches in Tree T

19 leaves in branch fDele
8 Branches of T

8 leaves of branch
Electrons

A double-click to histogram the leaf
The Tree Viewer & Analyzer

A very powerful class supporting complex cuts, event lists, 1-d, 2-d, 3-d views parallelism
Chains

Scenario:
Perform an analysis using multiple ROOT files. All files are of the same structure and have the same tree.

Chains are collections of chains or files

Chains can be built automatically by querying the run/file catalog
Chains of Trees

- A TChain is a collection of Trees.
- Same semantics for TChains and TTrees
  - `root > .x h1chain.C`
  - `root > chain.Process("h1analysis.C")`

```c
// creates a TChain to be used by the h1analysis.C class
// the symbol H1 must point to a directory where the H1 data sets
// have been installed

TChain chain("h42");
  chain.Add("$H1/dstarmb.root");
  chain.Add("$H1/dstarpla.root");
  chain.Add("$H1/dstarplb.root");
  chain.Add("$H1/dstarp2.root");
```
Tree Friends

```c
Root > TFile f1("tree1.root");
Root > tree.AddFriend("tree2","tree2.root")
Root > tree.AddFriend("tree3","tree3.root");
Root > tree.Draw("x:a","k<c");
Root > tree.Draw("x:tree2.x","sqrt(p)<b");
```

Processing time independent of the number of friends unlike table joins in RDBMS
The “No Shared Library” case

- There are many applications for which it does not make sense to read data without the code of the corresponding classes.
- In true OO, you want to exploit Data Hiding and rely on the functional interface.
- However, there are also cases where the functional interface is not necessary (PAW ntuples).
- It is nice to be able to browse any type of file without any code. May be you cannot do much, but it gives some confidence that you can always read your data sets.
- We have seen a religious debate on this subject.
- Our conclusion was that we had to support these two modes of operation.
- Support for the “No Shared Lib case” is non trivial
read/query Trees without the classes

```cpp
root [0] TFile f("Event.root")
Warning in <TClass::TClass>: no dictionary for class Event is available
Warning in <TClass::TClass>: no dictionary for class EventHeader is available
Warning in <TClass::TClass>: no dictionary for class Track is available
root [1] T_Show(45)
=====> EVENT:45
fUniqueID = 0
fBits = 50331648
fType[20] = 116 121 112 101 48 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fNtrack = 609
fNseq = 6068
fNvertex = 7
fFlag = 1
fTemperature = 20.529432
fEvtHdr.fEvtNum = 45
fEvtHdr.fRun = 200
fEvtHdr.fDate = 960312
fTracks = 609
fTracks.fPx = -0.677692, -2.625842, 1.130895, 3.095905, -0.209354, 0.988825, -1.232411, -1.119241, -0.580420, -0.378055
fTracks.fPy = 0.322117, 0.482258, -0.789722, -3
fTracks.fPz = 0.341093, 2.867670, 1.379341, 3.0
fTracks.fRandom = 529.431580, 525.431580, 529.431580, 529.431580
fTracks.fMass2 = 8.900000, 8.900000, 8.900000, 8.3
fTracks.fBx = -0.042621, 0.069631, -0.141984, 0
fTracks.fBy = 0.004728, 0.116303, 0.046655, -0
fTracks.fMeanCharge = 0.001209, 0.001738, 0.006828,
fTracks.fXFirst = 2.02409, 0.542266, -1.027804, -3
fTracks.fXlast = 5.743741, 15.938519, 0.260834, 14
fTracks.fYFirst = 10.330895, 2.485857, 16.579025, -7
fTracks.fYlast = -7.106707, 13.617641, 12.946399,
fTracks.fZFirst = 56.846382, 39.277451, 47.035370,
fTracks.fZlast = 211.758605, 213.753479, 217.62054,
fTracks.fCharge = 0.000000, 0.000000, 1.000000, 0.0
fTracks.fVertex[31] = -0.181014, 0.105711, -20.070364
0.034267, 0.06083, -3.763124, -0.119004, 0.038800, -1,
73879, -0.150654, 0.042276, 3.960069
fTracks.fNPoint = 64, 60, 65, 61, 62, 61, 62, 64, 6,
fTracks.fNValid = 1, 0, 1, 0, 1, 0, 1, 0, 1
fMeasure[101] = -2 0 5 2 5 1 14 13 9 11
fMatrix[4141] = -0.625079 0.530725 -0.796688 0.00
0.000000 0.000000
fClosestDistance = 1.44170 1.706780, -0.655489, 1.539576, 0.804130, 0.048241, -0.584909
```

root [2] new TBrowser
<class TBrowser*>0x88b7450
root [3]
Generate the classes header files
Compile them
make a shared lib
link the shared lib

```
MakeProject has generated 21 classes in xx
xx/MAKE file has been generated
Shared lib xx/xx.so has been generated
Shared lib xx/xx.so has been dynamically linked
root [5] ATLFElectron e
m_KFcode 11 Electron KF-code
m_MCParticle 6 Electron position in MCParticles list
m_KFmother 0 Electron mother KF-code
m_Eta 0 Eta coordinate
m_Phi 0 Phi coordinate
m_PT 0 Transverse energy
fUniqueID 0 object unique identifier
fBits 50331648 bit field status word
```
TFile::MakeProject

#ifdef ATLFElectron_h
#define ATLFElectron_h

#include " TObject.h"
#include " TAtt3D.h"

class ATLFElectron : public TObject, public TAtt3D {

public:
  Int_t  m_KFcode;    //Electron KF-code
  Int_t  m_MCParticle; //Electron position in MCParticles list
  Int_t  m_KFmother;  //Electron mother KF-code
  Float_t m_Eta;      //Eta coordinate
  Float_t m_Phi;      //Phi coordinate
  Float_t m_PT;       //Transverse energy

  ATLFElectron() {};
  virtual ~ATLFElectron() {};

  ClassDef(ATLFElectron,1) //
};

ClassImp(ATLFElectron)
#endif
ROOT Folders
Why Folders?

This diagram shows a system without folders. The objects have pointers to each other to access each other's data.

Pointers are an efficient way to share data between classes. However, a direct pointer creates a direct coupling between classes.

This design can become a very tangled web of dependencies in a system with a large number of classes.
Why Folders?

In the diagram below, a reference to the data is in the folder and the consumers refer to the folder rather than each other to access the data.

The naming and search service provided by the ROOT folders hierarchy provides an alternative. It loosely couples the classes and greatly enhances I/O operations.

In this way, folders separate the data from the algorithms and greatly improve the modularity of an application by minimizing the class dependencies.
Posting Data to a Folder

(Producer)

- No changes required in user class structure.
- Build a folder structure with:
  - `TFolder::AddFolder(TFolder *)`
- Post objects or collections to a Folder with:
  - `TFolder::Add(TObject*)`
- A TFolder can contain other folders or any TObject descendents. In general, users will not post a single object to a folder, they will store a collection or multiple collections in a folder. For example, to add an array to a folder:
  - `TObjArray *array;`
  - `run_mc->Add(array);`
Reading Data from a Folder (Consumer)

One can search for a folder or an object in a folder using the `TROOT::FindObjectAny` method. `FindObjectAny` analyzes the string passed as its argument and searches in the hierarchy until it finds an object or folder matching the name. With `FindObjectAny`, you can give the full path name, or the name of the folder. If only the name of the folder is given, it will return the first instance of that name.

```cpp
conf = (TFolder*)gROOT->FindObjectAny("/alroot/Run/Configuration");
```

or

```cpp
conf = (TFolder*)gROOT->FindObjectAny("Configuration");
```

A string-based search is time consuming. If the retrieved object is used frequently or inside a loop, you should save a pointer to the object as a class data member. Use the naming service only in the initialization of the consumer class.
Example: Alice folders

Some of the AliRoot folders shown in the browser:

A ROOT Tree can be automatically generated from the folder, eg:

```cpp
TTree T("T","/Event");
T.Fill();
```
ROOT working with Objectivity

http://www.phenix.bnl.gov/WWW/publish/onuchin/rooObjy/

Introduction

This is library of ROOT wrappers around Objectivity base classes and functions.

- **The Goals:**
  - Create a tool library which will allow the retrieval of data from Objy DB from a ROOT prompt.
  - Provide the basis library for ROOT GUI PHENIX Database Browser
  - The library was developed with hope it can be used inside PHENIX software framework

- **The Status:**
  - More than 40 Objectivity classes were wrapped with 10k lines of code and filled with Objy docs. Including:
    - Objectivity object handler and iterator classes, i.e. ooHandle(XXX), ooIter(XXX)
    - Objectivity Active Schema classes
    - Objectivity global functions, constants etc.
  - Compiled on Linux. Tested ++ more testing required. Not ported to S

- **The library allows users to:**
  - connect to Federated database
  - open up database
  - iterate through databases/containers
  - obtain descriptions of the classes in database
  - retrieve persistent data for any object in the database
  - ++ much more, interactively from ROOT

---

If you have any comments about this page please send them to Valery Onuchin
ROOT working with Oracle


See also: http://www.gsi.de/computing/root/OracleAccess.htm

Introduction

- This library is a set of ROOT wrappers of libodbc++ classes that provides an interface based on the JDBC API.
- It corresponds to JDBC-ODBC bridge in Java terminology and makes it possible to do:
  1. establish a connection from ROOT session to any database for which ODBC driver available.
  2. send SQL statements.
  3. process the results.

- Major differences between libodbc++ and RDBC:
  - RDBC written according to ROOT coding conventions. All method names begin with upper case letter.
  - Since ROOT C++ Interpreter (CINT) does not provide full support of namespaces, all RDBC classes have TSQL prefix, e.g. TSQLStatement corresponds to odbc::Statement etc.
  - Since ROOT C++ Interpreter (CINT) does not provide full support of exceptions, R4 object communication mechanism is used for exception handling (see TSQL::SetHandler method).
  - RDBC supports
    - TSQLResultSet::GetColumn
    - TSQLResultSet::UpdateObject
    - TSQLPreparedStatement::SetObject

- We consider RDBC as a low-level API and a base for higher-level interface between ROOT based offline environment of MINOS experiment and Oracle database. New ROOT-RDBC based GUI tools for Oracle are coming. Stay tuned!
Time to conclude

We have a working system
Used by many people
In many different configurations

but
General remarks

- In 1995, we had planned less than 50% of ROOT 2001.
  - importance of dictionary, RTTI
  - Automatic Schema Evolution
  - effort in GUI
  - Online requirements (Threads, Timers, Sockets, etc)

- Development of a system is driven by:
  - ideas from authors
  - ideas from users
  - new ideas and techniques in computing
  - OS development. In 1995, push for Windows, Linux not here
  - language developments (eg template support, exception handling, Java)
  - cooperation with other systems (ex Objy, Oracle, Corba, Qt, etc)
  - manpower

Users expect stable and working systems. Quality of a system should improve with time. Often in contradiction with major developments.
ROOT: an Evolving System

- The ROOT system has been in continuous development since 1995 surviving major changes, major enhancements and an ever increasing number of users.
- In the same way that Root2001 is far from the original Root1995, we expect that Root2006 will include many contributions reflecting the continuous changes and new ideas in the field of computing.
- This implies a strong cooperation between software developers in the major experiments.
- Root is being developed in very close cooperation with a cloud of software developers in small, medium and large experiments. Computer scientists from non-HEP fields are also contributing.
Download source, Binaries

http://root.cern.ch

- Intel x86 Linux for Redhat 7.1 (glibc 2.2) and gcc 2.96, version 3.02/01 (3.1 MB).
- Intel x86 Linux for Redhat 6.1 (glibc 2.1) and gcc 2.95.2, version 3.02/01 (9.7 MB).
- Intel x86 Linux for Redhat 6.1 (glibc 2.1) and egcs 1.12, version 3.02/01 (8.2 MB).
- Intel x86 Linux for Redhat 5.0/5.1/5.2 (glibc) and egcs 1.11, version 3.02/01 (8.6 MB).
- Intel Itanium Linux for Redhat 7.0 (glibc 2.2) and gcc 2.96, version 3.00/05 (9.0 MB).
- HP-UX 10.20 with aCC (v1.18), version 3.02/01 (12.7 MB).
- Compaq Alpha OSF1 with cc 6.2, version 3.02/01 (9.3 MB).
- Compaq Alpha OSF1 with egcs 1.12, version 3.02/01 (10.6 MB).
- Compaq Alpha Linux with egcs 1.12, version 3.01/06 (11.0 MB).
- Compaq iPAQ PocketPC Linux with gcc 2.95, version 3.00/06 (6.7 MB).

For more on Linux on iPAQ see www.handhelds.org.

- AIX 4.3 with z/CE version 3, version 3.02/01 (11.2 MB, works only on AIX 4.3).
- AIX 4.5 with z/CE version 5, version 3.02/01 (11.3 MB, works only on AIX 4.5).

- Sun SPARC Solaris 5.6 with CC 2.2, version 3.02/01 (8.5 MB).

It cannot be used with Solaris 5.7 or 5.8 even using the same compiler version. You must recompile from the source on these two systems.

- Sun SPARC Solaris 5.7 with CC 2.0, version 3.02/01 (9.7 MB).

It cannot be used with Solaris 5.6 or 5.8 even using the same compiler version. You must recompile from the source on these two systems.

- Sun SPARC Solaris 5.8 with CC 2.2, version 3.02/01 (10.1 MB).

It cannot be used with Solaris 5.6 or 5.7 even using the same compiler version. You must recompile from the source on these two systems.

- SGI IRIX 6.5 with CC, version 3.02/01 (compiled with -n32) (10.5 MB).
- SGI IRIX 6.5 with g++ 2.95.2, version 3.02/01 (11.6 MB).
- SGI IRIX 6.5 with GCC, version 3.02/01 (10.1 MB).

- Linux PPC/2000 (glibc 2.1) gcc 2.95, version 3.01/05 (7.8 MB).

Thanks to Danne Buskolic (buskolic@ipp.in2p3.fr) for building this version.

- Windows/NT 95/98 with VC++ 6.0, version 3.02/01 (good old tar file) (9.3 MB).
- Windows/NT 95/98 with VC++ 6.0, compiled with debug info, version 3.02/01 (good old tar file) (17.4 MB).
- Windows/NT 95/98 with VC++ 6.0, version 3.02/01 (built with InstallShield) (9.1 MB).

When running from the MSDOS prompt, you must set the following environment variables, eg in your autoexec.bat:
(Restart the system if you set these variables for the first time).
Makefiles

- 3 major OS (Unix, Windows, Mac OS/X)
- 10 different compilers
  - gcc with many flavors on nearly all platforms,
  - Solaris: CC4,5, HPUX: CC:aCC, SGI: CC, AIX: xlc
  - Alpha: CXX6, Windows: VC++6
  - KAI on SGI, Linux, Solaris
- 37 Makefiles

```bash
(pcnotebrun) [732] ls "/root/config
ARCHS      Makefile.freebsd4 Makefile.linuxdebo2 Makefile.linuxsuse6 Makefile.solarisCC5
CVS        Makefile.hpux      Makefile.linuxdebo2ppc Makefile.lynvos    Makefile.solarisgecs
Makefile.aix Makefile.hpuxacc Makefile.linuxegcs  Makefile.macosx     Makefile.solarisgcc
Makefile.aixegcs Makefile.hpuxegcs Makefile.linuxia64gcc Makefile.mklinux    Makefile.solariskcc
Makefile.alphacxx6 Makefile.in    Makefile.linuxia64sgi Makefile.sgiccc    Makefile.win32
Makefile.alphagecs Makefile.linux     Makefile.linuxkcc    Makefile.sgiegcs    config.in
Makefile.alphakcc Makefile.linuxalphaegcs Makefile.linuxpgcc Makefile.sgilcc    root-config.in
Makefile.config Makefile.linuxarm    Makefile.linuxppegcs Makefile.sgin32egcs  rootrc.in
Makefile.freebsd Makefile.linuxdeb Makefile.linuxxrh42   Makefile.solaris       
```
ROOT Downloads

- 129,000 binaries download
- 650,000 clicks per month
- 30,000 docs in 12 months
- 2200 reg users in roottalk

Total FTP distributions: 129030

(ROOT binaries only)
# ROOT Users in the large experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Registered Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLAS</td>
<td>133</td>
</tr>
<tr>
<td>ALICE</td>
<td>120</td>
</tr>
<tr>
<td>CDF</td>
<td>88</td>
</tr>
<tr>
<td>PHENIX</td>
<td>86</td>
</tr>
<tr>
<td>CMS</td>
<td>85</td>
</tr>
<tr>
<td>STAR</td>
<td>82</td>
</tr>
<tr>
<td>JLAB</td>
<td>77</td>
</tr>
<tr>
<td>D0</td>
<td>70</td>
</tr>
<tr>
<td>BABAR</td>
<td>69</td>
</tr>
<tr>
<td>H1</td>
<td>48</td>
</tr>
<tr>
<td>L3</td>
<td>43</td>
</tr>
<tr>
<td>HERAB</td>
<td>37</td>
</tr>
<tr>
<td>NA49</td>
<td>37</td>
</tr>
<tr>
<td>LHCB</td>
<td>35</td>
</tr>
<tr>
<td>DELPHI</td>
<td>34</td>
</tr>
<tr>
<td>ZEUS</td>
<td>32</td>
</tr>
<tr>
<td>HADES</td>
<td>27</td>
</tr>
<tr>
<td>PHOBOS</td>
<td>27</td>
</tr>
<tr>
<td>WA98</td>
<td>25</td>
</tr>
<tr>
<td>BELLE</td>
<td>22</td>
</tr>
<tr>
<td>COMPASS</td>
<td>22</td>
</tr>
<tr>
<td>KLOE</td>
<td>19</td>
</tr>
<tr>
<td>ALEPH</td>
<td>18</td>
</tr>
<tr>
<td>OPAL</td>
<td>17</td>
</tr>
<tr>
<td>AUGER</td>
<td>16</td>
</tr>
<tr>
<td>MINOS</td>
<td>16</td>
</tr>
<tr>
<td>NOMAD</td>
<td>16</td>
</tr>
<tr>
<td>BRAHMS</td>
<td>15</td>
</tr>
<tr>
<td>GLAST</td>
<td>14</td>
</tr>
<tr>
<td>AMS</td>
<td>12</td>
</tr>
<tr>
<td>NA45</td>
<td>12</td>
</tr>
<tr>
<td>NA48</td>
<td>11</td>
</tr>
<tr>
<td>AMANDA</td>
<td>10</td>
</tr>
</tbody>
</table>

Registered users in the ROOT system
ROOT team today
ROOT Educational Resources at FNAL

http://www-pat.fnal.gov/root/

Day 1: slides, Power Point file
- Overview of the ROOT Framework
- GUI basics
- Command line basics
- Finding Information

Day 2: slides, Power Point file
- Command Line (CINT)
- Scripts (CINT & ACLiC)
- Getting started with the
- Exercises Functions and Fitting
- TreeViewer

Day 3: slides, Power Point file
- Building Root Trees
- Reading Root Trees
- Using Trees in Analysis
- Add your class to ROOT

Examples: download all the example files used in the tutorial

Exercises: download the exercises we use in class, they are the same as we used at CERN school. Here is a list of the things that you will learn:
- three ways you can use ROOT: the command line, the script processor, and the graphical user interface (GUI).
- how to generate a PostScript file.
- histograms, and the input/output capabilities.
- example of an analysis using real physics data.
- example of a simulations and an event display.

C++ Basics for ROOT users
Follow this link for a refresher on C++.

ROOT User’s Guide
Follow this link to the preliminary version of the ROOT User’s Guide.
Hard copies of the ROOT User’s Guide are now available in the Fermilab Stock Room.
The part number is: 1307-058000 and the cost is $5.21 a copy.

ROOT Mailing Lists
Follow this link to find the mailing lists and archives for roottalk and about root. This is a place where you can ask questions, and get an answer quickly. You can also scan the archive to find topics of interest.

Fermilab’s ROOT Releases
Follow this link to see a list of ROOT releases available at Fermilab.