ROOT in the Minos Experiment

Presented at ROOT2002 (CERN)

October 14, 2002

George Irwin - Stanford University

Contributions from:

Robert Hatcher - FNAL

Susan Kasahara - U. of Minnesota

David Petit - U. of Minnesota

Brett Viren - Brookhaven Lab

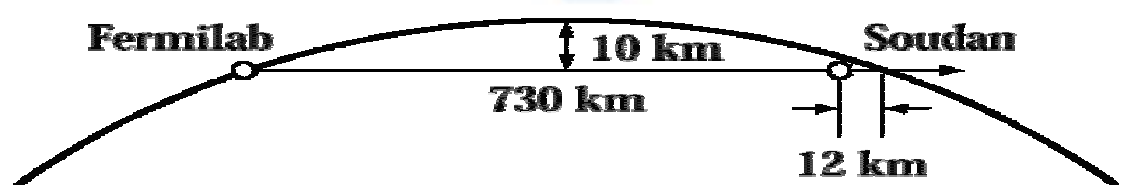
Nick West - Oxford University

Minos - a long baseline neutrino oscillation experiment

3 Detectors

1. Near Detector (ND) at Fermilab
2. Far Detector (FD) at Soudan, Minnesota
3. Calibration Detector (CalDet) at CERN

Unified software system must be modular and dynamically configurable.



Minos Status

Hardware

- CalDet taking data in CERN test beam
- Far Detector is 2/3 completed - taking cosmic data
- Near detector beginning construction
- Neutrino beam due from Fermilab in 2005
- Planning for a 20 year experiment lifetime

Software

- ROOT-based reconstruction framework analysing data from CalDet and Far Detector
- MC/Sim framework to be rewritten in C++

Remainder of Talk

Highlights a few special ways that Minos uses ROOT:

- Data Model
- Data Dispatcher and its clients
- Database Interface
- TG wrappers

Minos: Data Model and I/O

Overview

- Extensive use of ROOT persistency tools: Streaming mechanism, File management, Data structures (TTree), and Remote data access (both TNetFile and TSocket).

- Data is organized into data streams. A stream is a TTree containing objects of a single class type extending over 1 or more sequential files. Framework supported streams:
  - DAQ data streams:
    - Raw event records
    - Calibration records
    - DAQ monitor records
  - Reconstructed event records
  - Ntuple (or event summary) records derived from reconstruction data (under development)
  - Monte Carlo records

  A user may also configure their own data stream at the JobControl script command line.

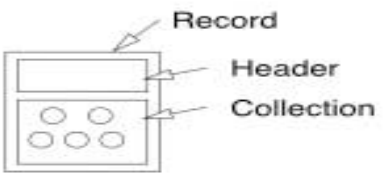
- All record types derive from a common record base class and have a header deriving from a common header base class. The minimum data content of the record header is the event VldContext (event date, time and detector type) used to associate records across the different data streams.

Records and Streams

Record : **a self–contained unit of data, usually written after some specific state–transition**
(e.g. RawDaqFrame for each trigger, RawDaqLISummary for light injection run)

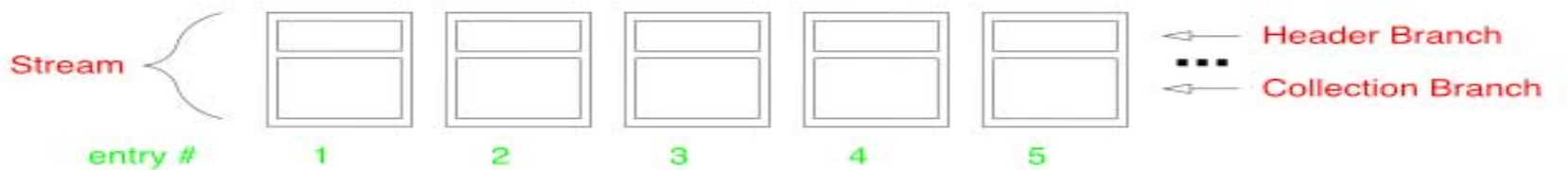
Header : **small block of identifying information**
All records have: timestamp; {Near,Far,CalDet}; {Data,MC}
Record type specific info, e.g. RawDaqFrame has frame # and trigger bits

Collection : **a heterogenous aggregation of objects**



Stream : **an ordered sequence of Record s**
expressed in ROOT file as TTree object

Branch : **an independently written (read) unit**
Minos Offline data model has two branches: header and collection



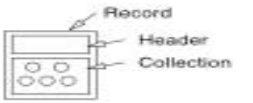
Records and Streams

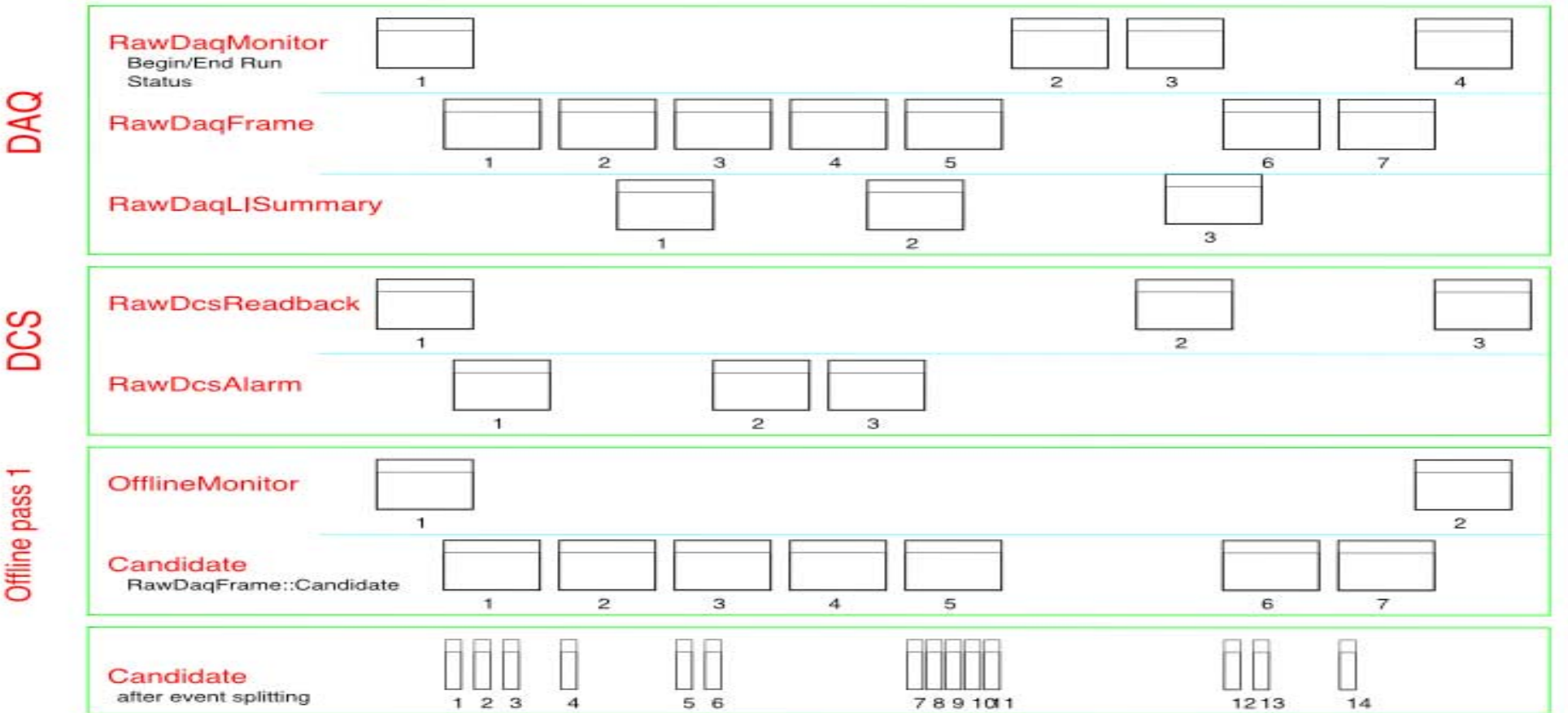
TRef links between Inter-Stream Records

Records are written asynchronously for each Stream

Offline pass 1 Candidate Records have a 1:1 match to RawDaqFrame Records

Other possible Streams: SimulationMonitor and SimulationFrame

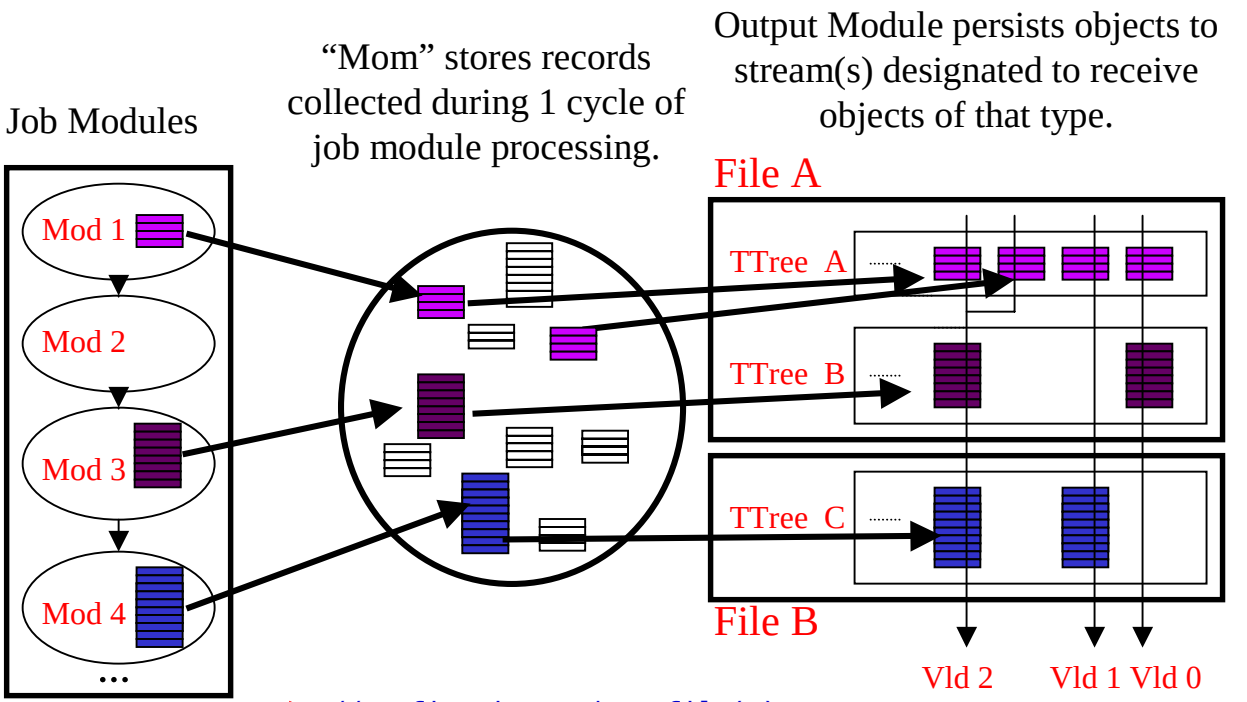




Minos: Data Model and I/O

On output, streams are created to persist objects of a specific record type.

```
{
 JobC j;
 …
 // Stream “MyStream” is defined to persist
 // records “MyRecord”. Internally, tree
 // “MyStream” is created with a single
 // branch “MyRecord”, split at level=99.
 j.Path(“Demo”).Mod(“Output”).
     Cmd(“DefineStream MyStream MyRecord”);

 // Enable list of output streams for this job
 j.Path(“Demo”).Mod(“Output”).Set(“Streams
                                =MyStream,…”);

 // Direct the trees to an output file. Trees
 // may be directed to different output files.
 j.Path(“Demo”.Mod(“Output”).Set(“FileName
                                 =demo.root”);
 …
 }
```



On input, records from multiple input streams are sequenced by record VldContext. Records of a common VldContext are loaded into “Mom” as a single record set.

```
 {
  JobC j;
  …
  // Enable list of input streams.
      j.Input.Set(“Streams=MyStream,…”);
```

```
// Define input data file(s)
j.Input.AddFile(“demo1.root”);
j.Input.AddFile(“demo2.root”);

// The user may apply a selection cut.  This cut is
// applied internally using the TTreeFormula class
// (as in TTree::Draw), and data from required
// branches only is read in to apply cut.
j.Input.Select(“MyStream”,
                ”((MyHeader*)fHeader)->GetNTrack()>2”);
…
}
```

Minos: Data Model and I/O

Recently, we have been working to remodel the record package and to design an Ntuple record class.

The new record base record class makes use of a templated data member:

```
template <class T> class RecRecordImp: public RecRecord { // RecRecord is abc class inheriting from TNamed
 …
 T fHeader;
};
```

Tests show that we can successfully I/O records from a ROOT TTree created with split level=99 to hold objects of this templated class type.

```
RecRecordImp<RecHeader> *record = 0;
fTTree->Branch("RecRecordImp<RecHeader>","RecRecordImp<RecHeader>",&record, 64000,99);
```

The Ntuple record class makes use of TClonesArrays. It would be useful to have more flexibility in how these can be used. In particular, TClonesArrays in a base class do not split properly when a derived class is used to create the main branch of the TTree.

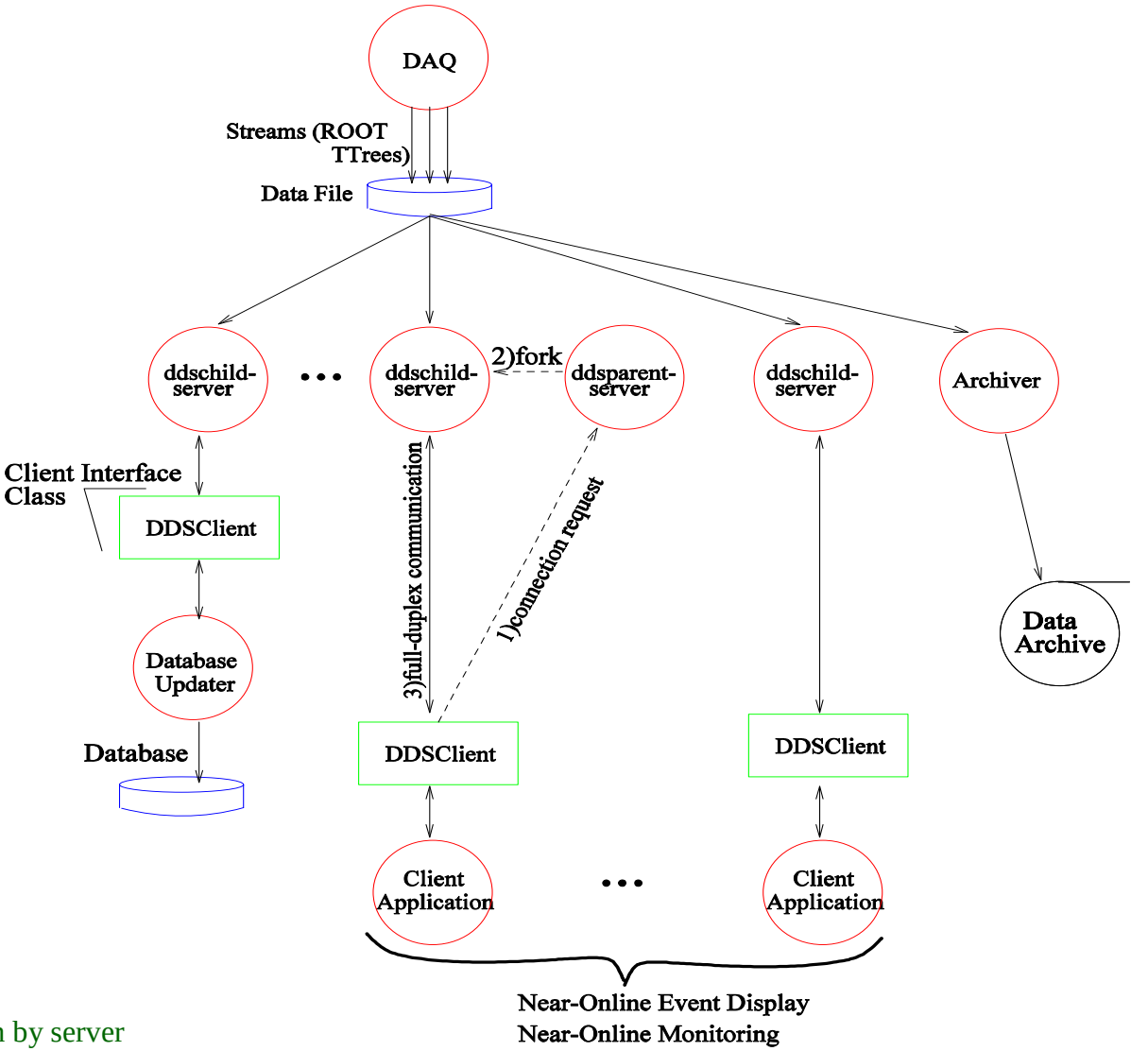
```
class Event: public TObject {
 …
 TClonesArray* array;
};
class SpecialEvent: public Event{
 …
};

SpecialEvent* event = 0;
fTTree->Branch("SpecialEvent","SpecialEvent",&event,64000,99); // fails to split base class TClonesArray
```

Minos: Data Model and I/O (Dispatcher)

The Data Dispatcher serves data from the DAQ generated output file to near-online clients (local and remote). The data is served before the data file has been closed by the DAQ.

Some features of the system:

- The read of an open ROOT data file is accomplished without the use of file locks. The Daq regularly saves the TTree to file and updates the TDirectory keys, and the reader checks these keys as needed for the availability of a new TTree. When a collision between the writer and reader occurs, the reader recognizes that it has a corrupt data buffer, signaled by an error in ROOT's internal unzipping method, and tries again.
- The servers (parent and child) make use of ROOT's TSocket to communicate with clients.
- The client may subscribe to certain subsets of data and the client's selection criteria will be applied server side.




```
{
 JobC j;
 j.Input.Set("Format=DDS");
 j.Input.Set("DDSServer=ddsserver.host.address");
 j.Input.Set("Stream=DaqSnarl"); // this tree only is read in by server
 j.Input.Select("DaqSnarl","((RawDaqSnarlHeader*)fHeader)->GetTrigSrc()==2");  // applied on server side via TTreeFormula
 …
}
```

Minos: Online Monitoring - Principles

Package to check data quality and detector performance in real-time.

Runs automatically at the CERN and Soudan detector sites.

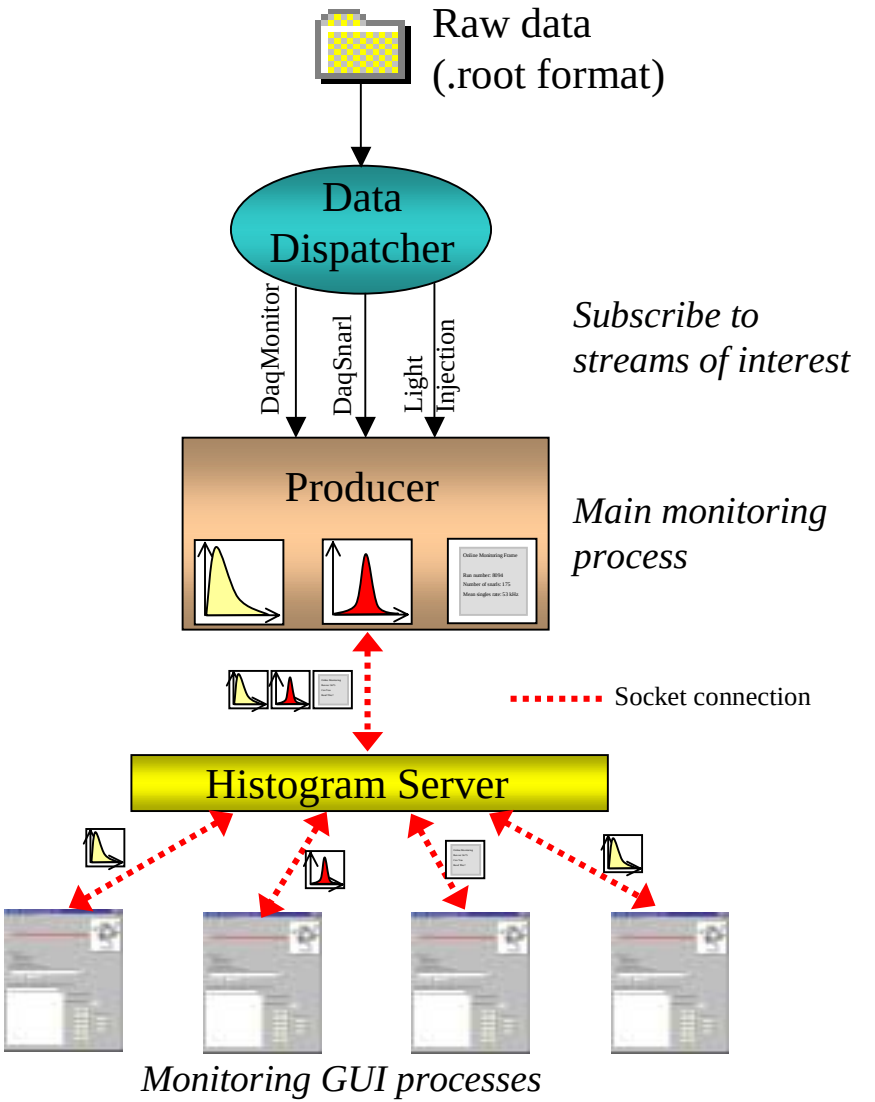
Based on CDF RunII Online Monitoring framework (see H. Stadie talk in ROOT2001).

Consists of three processes:

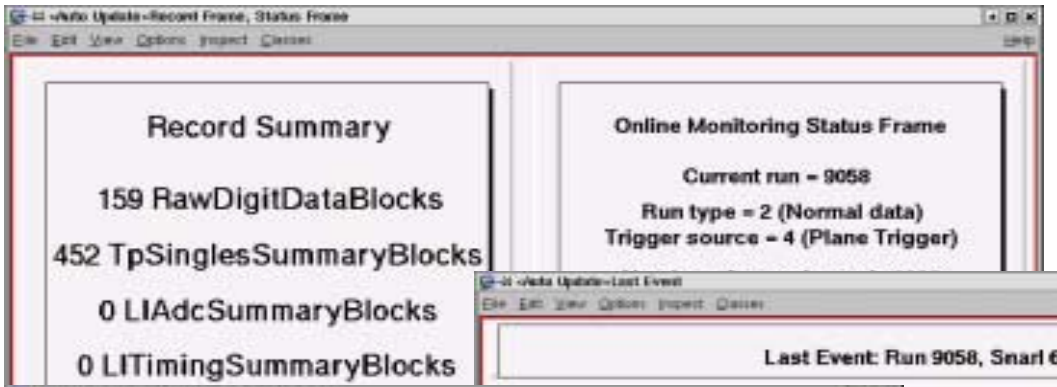
- Producer: receives and analyses data from the Data Dispatcher. Uses MINOS C++ analysis framework.
- Histogram Server: receives ROOT histograms from the Producer via socket connection.
- ROOT-based GUI: connects to Histogram Server. Handles histogram plotting and updates.

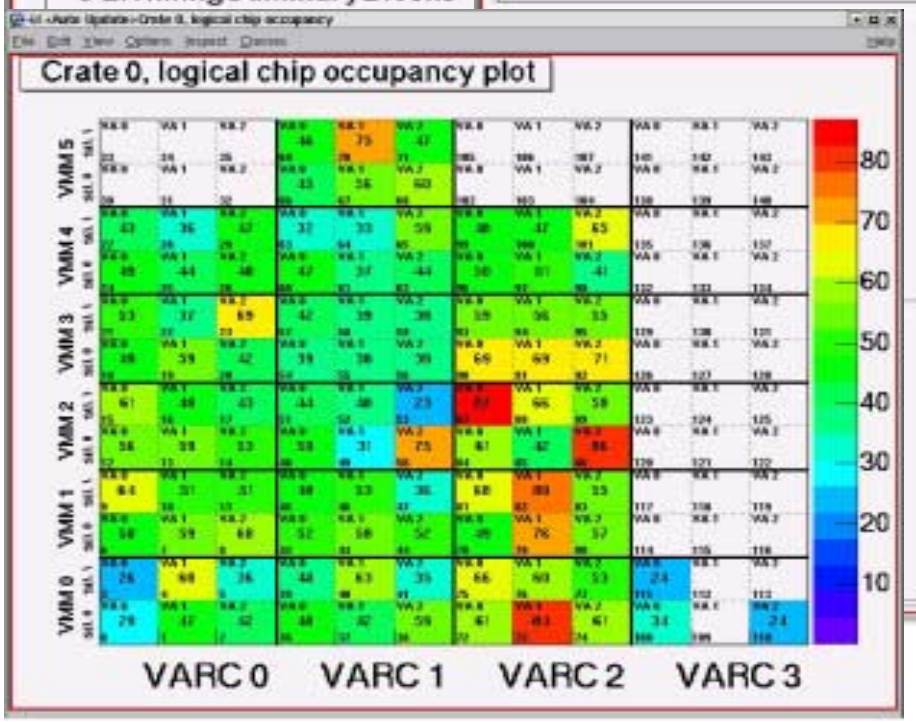
GUI is decoupled from Producer/Server:

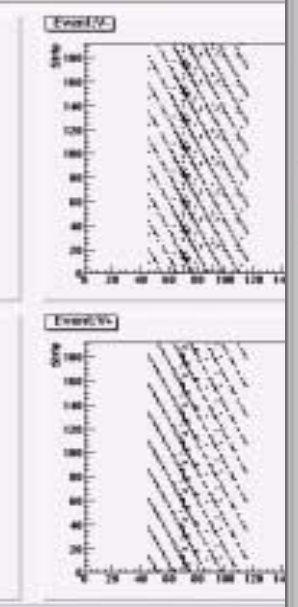
- several GUIs running at external institutions can connect to a single Producer (e.g. at Soudan) and monitor the status of the detector



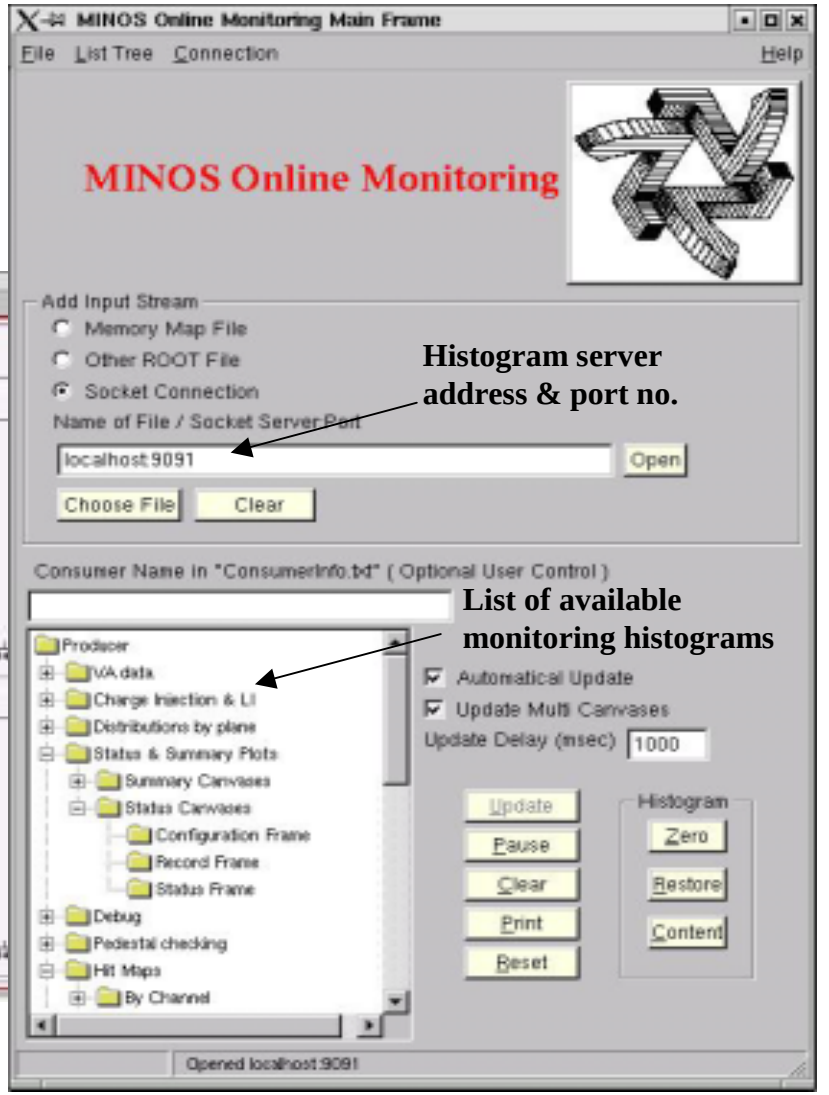
Minos: Online Monitoring – Sample output







Sample monitoring canvases



Online Monitoring GUI

Minos: Database Interface: Requirements

Purpose

- Provide detector configuration (Geometry, Cabling, Calibration) for event reconstruction.
- Provide standard software configurations (Cuts, Switches) for production jobs.

Concepts

- *Context*: event date, time and detector (class `VldContext`).
- *Range*: A *Context* extended to a time window (class `VldRange`).
- *TableRow*: A single row of a table (sub-class of `DbiTableRow`).
- *Aggregate*: A set of *TableRows* sharing a *Range*.

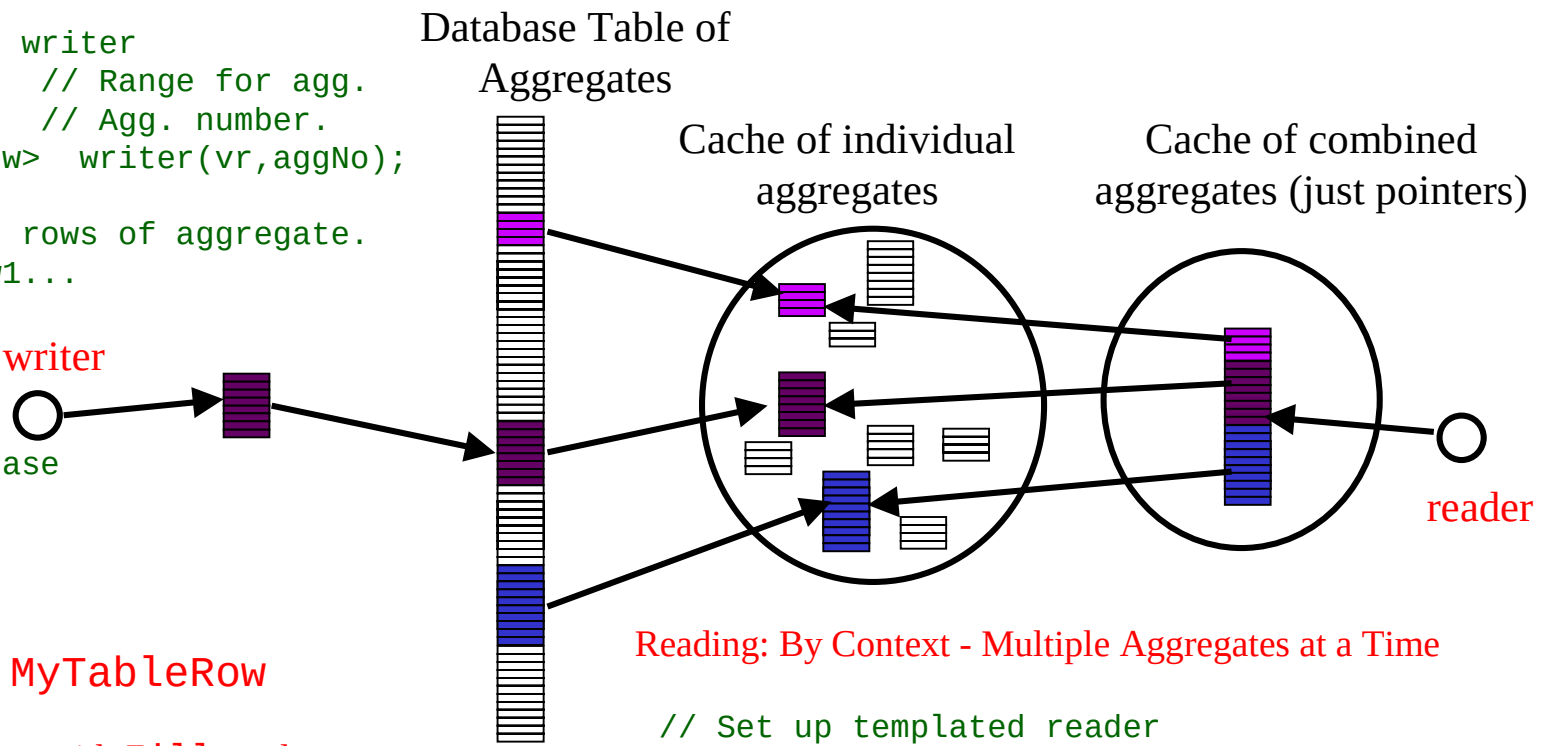
Principle Requirements

- Write Access by Range
  - Use Case: "*Calibrate every channel in this crate, estimate Range for which it remains valid and store in database*."
- Read Access by Context
  - Use Case: "*For this event (context) get calibration constants for every channel in every crate*."
  - Must be efficient (so have Cache that owns query results for reuse, caller just gets const pointer)
- Database Distribution
  - Use Case: "*Copy data from source (detector) databases to local mirror databases*."
  - Detectors configurations from FNAL (Near det.), Soudan (Far det.), CERN (Calib. det).
  - Store in Master Database and mirror to local databases.
  - Exchange data at aggregate level between databases, validating exchanges.

Minos: Database Interface: Access

Writing: By Range - One Aggregate at a Time

```
// Set up templated writer
const VldRange& vr;  // Range for agg.
Int_t aggNo;         // Agg. number.
DbiWriter<MyTableRow>  writer(vr,aggNo);

// Fill writer with rows of aggregate.
MyTableRow row0,row1...
writer << row0;
writer << row1;
...

//  Commit to Database
writer.Close();
```



Adding New Tables e.g. `MyTableRow`

Inherit from `DbiTableRow` with `Fill` and `Store` methods e.g.

```
void MyTableRow::Fill(DbiResultSet& rs)
{ rs >> fMember1 >> fMember2 >> ...;}
```

Other Methods to Get at Data e.g.

```
Float_t GetGain() const { return fMember1; }
```

Reading: By Context - Multiple Aggregates at a Time

```
// Set up templated reader
VldContext vc;  // Context from event
DbiResultPtr<MyTableRow> reader(vc);

// Random access to table rows
const MyTableRow* row0 = reader.GetRow(0);
const MyTableRow* row1 = reader.GetRow(1);
```

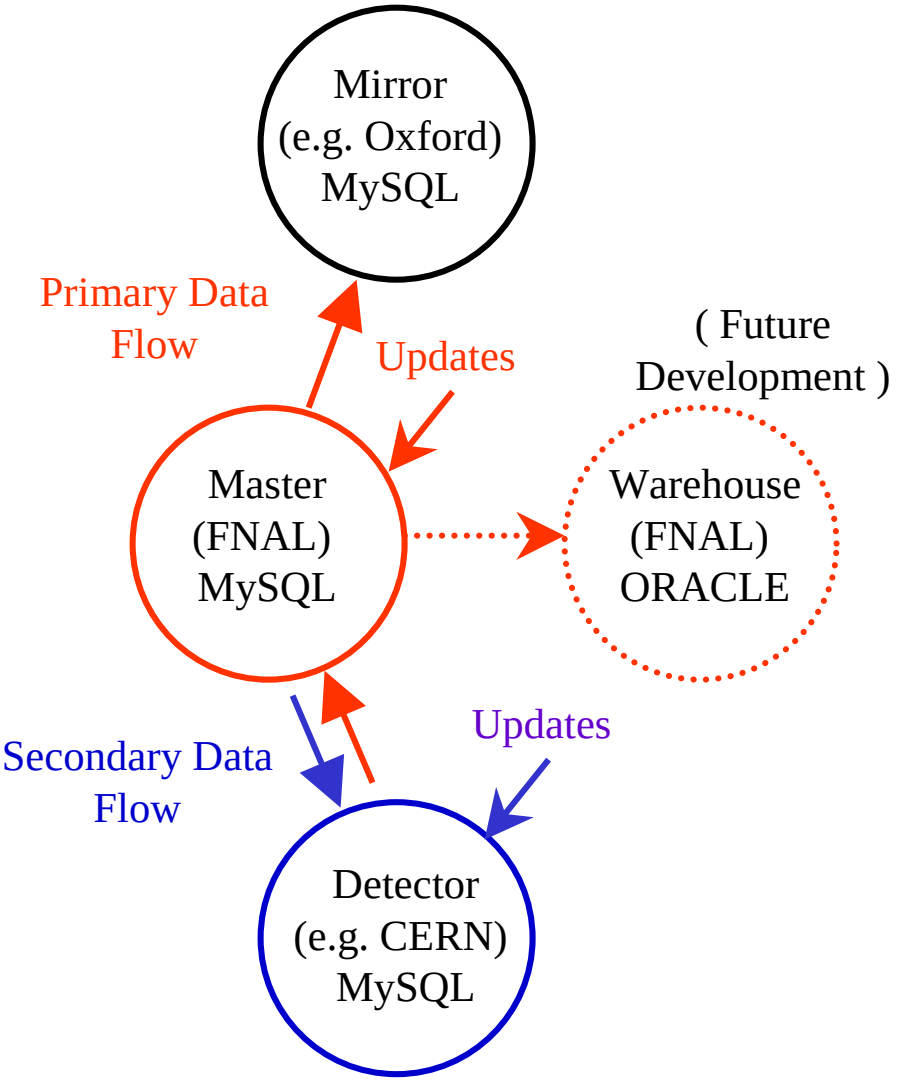
Minos: Database Interface: Distribution

Databases & Data Flows

- *Detector*: Source of configuration data.
- *Master*: Primary source.
- *Mirror*: Used locally for standard production.
- *Warehouse*: (eventually) ORACLE as robust permanent store for all data.

The unit of update is an Aggregate, having:-

- Unique Sequence Number
  - To prevent duplication (either ignore or replace if same Sequence Number).
- Local Insert Date
  - Identify changes when exporting updates.
  - Roll-back (ignore Insert > roll-back date).
- Creation Date
  - Identify replacements (replace if later).
  - Validation: Export twice. On import compare and report differences.



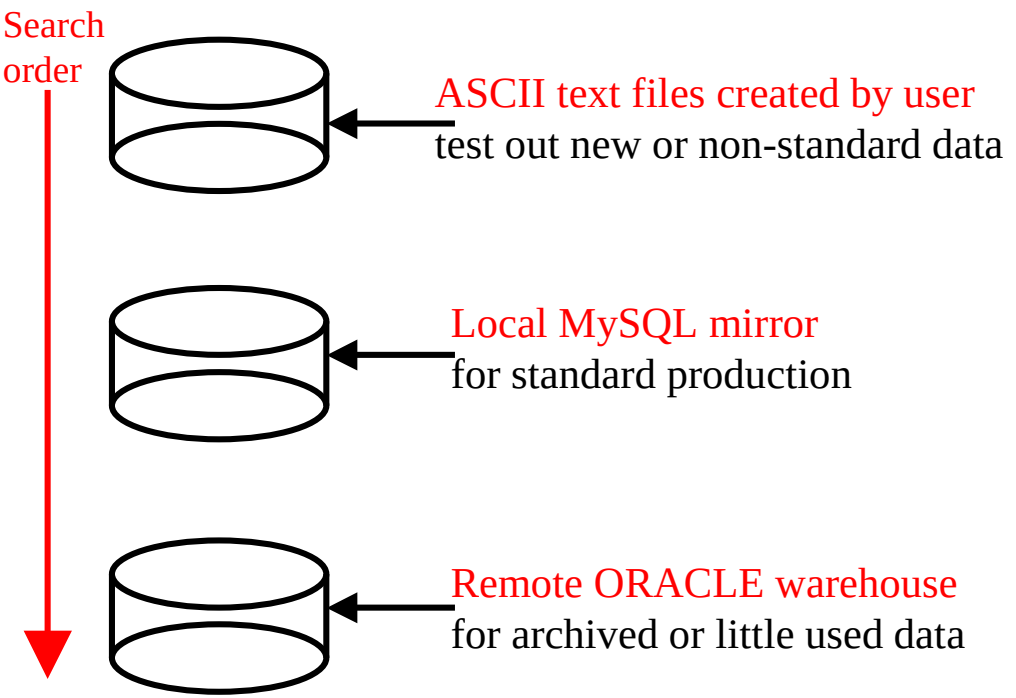
Minos: Database Interface: Local Configuration

A “Cascade” of Databases
- A priority-ordered list of databases.
- Upper ones overlay (hide) lower ones.

Cascade configured at Execution Time
- To try out alternatives.

Heterogeneous DB Technologies
- MySQL.
- ORACLE.
- Text files (but currently only via MySQL client creating temporary tables).
- Eventually possibly Postgres?

Relies on ODBC Interface
- to support heterogeneous DB technologies.



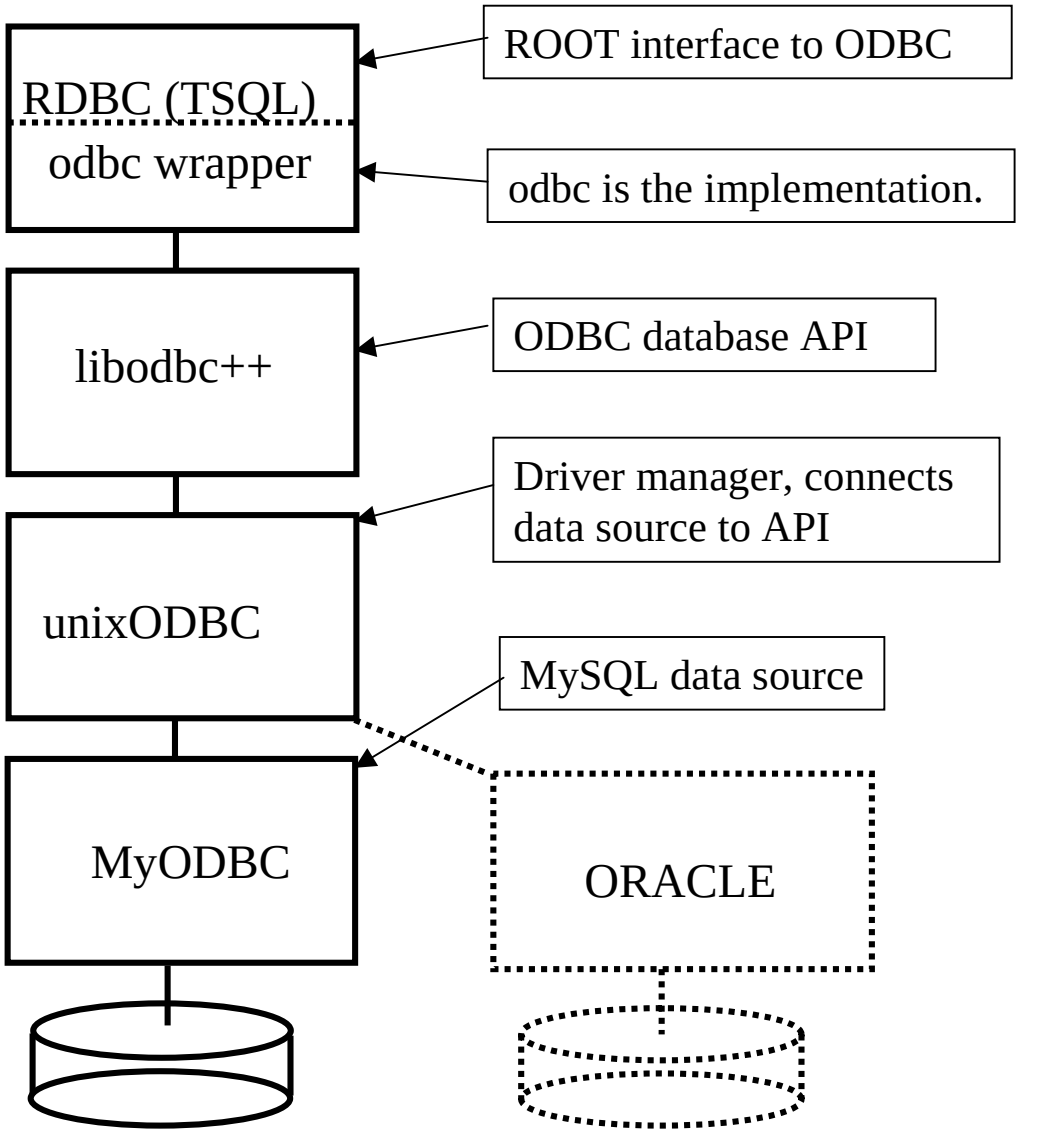
Minos: Database Interface: The ROOT Connection

Minos Needs ODBC

- DBI talks to database products through RDBC (TSQL), Valeriy Onuchin's extension to ROOT's TSQL.
- Dennis Box (FNAL) is extending odbc wrapper (from omanush@stendahls.net) and driver (from EasySoft) for ORACLE, see:-
  http://fndapl.fnal.gov/~dbox/oracle/odbc/
- From there use public domain layers to DB backend.

RDBC implements a generic “ODBC" interface to database products

Minos would like ROOT to adopt and distribute the RDBC extension to TSQL



ROOT for Gui and Event Display

BROOKHAVEN NATIONAL LABORATORY

- Wrapping of TG classes
  - Why do it
  - How it is done
- Example use of MVC: Ranges
- ROOT Improvement Wishlist

Why Wrap TG Classes

BROOKHAVEN NATIONAL LABORATORY

- Attempt to somewhat simplify API at the possible expense of access all TG features.
- TG has no widget memory management protocol. Must do memory management by hand and external to the widgets.
- Creating/managing TGLayoutHints awkward. Why create an object just to pass in a bitfield and a few config params?
- Rt (signal/slot) inadequate. Run-time connections lead difficult to find run-time bugs. Lack of support for a range of signal interfaces. Arbitrary data passing only possible kludgey tricks.. Reliance on CINT for signal emission. Slots not first class objects.

Gui classes: How TG Classes Are Wrapped

BROOKHAVEN NATIONAL LABORATORY

- ROOT team makes all Rt `*SIGNAL*` methods virtual. Big Thanks!
- Inheriting classes override these methods to emit a libsigc++ signal (Rt ones still emitted by calling parent class's method).
- All child widgets passed to parent by reference.
- All child widgets held by address in a `SigC::Ptr<>` ref counted pointer.
- Zero ref count in `SigC::Ptr<>` triggers deletion. Reference counting only turned on if widget created via `SigC::manage(new GuiSomeWidget)`. Normal heap or stack creation means child not managed - can mix an match in a single parent.
- `GuiBase` base class handles memory management of TGLayoutHints and maintains `list<SigC::Ptr<GuiBase> >` of children.

Eg of MVC pattern: Ranges

BROOKHAVEN NATIONAL LABORATORY

Model:

- `template<class T> class Range` maintains min and max values.
- Provides Set/Get methods, emits `modified` libsigc++ signal on Set()s.

View:

- Any object can attach to a Range's signal to respond to a change.

Control:

- Don't modify Range directly, but go through intermediate: `template<class T> class RangeControl`
- Extra functionality: `RangeControl` IsA `Undoable`. Before each set, puts Range's state (in form of libsigc++ Slot object) into an `UndoHistory`.
- Get undo/redo functionality "for free". No new code in Views needed for this functionality.

Brett Viren, BNL - Use of ROOT in MINOS, 14 Oct, 2002 – p.4/

GuiSlider: Example Case.

BROOKHAVEN NATIONAL LABORATORY

- `class GuiSlider` HasA `TGDoubleSlider` and two `Range<double>`s, one for extrema and one for current min/max
- Attach to range:

```
void print_range(Range<double>* r) {
  cout << r->Min() << "," << r->Max() << endl;
}
// ...
using namespace SigC;
GuiMainWindow mw(100,20)
GuiSlider gs(mw,kHorizontalFrame);
mw.Add(gs);
UndoHistory* hist = manage(new UndoHistory);
Range<double>* r = manage(new Range<double>);
RangeControl<double> rc(hist,r);
gs.UseRangeControl(rc);
r.modified.connect(bind(slot(print_range),&gs));
mw.ShowAll();
```

ROOT (and CINT) Feature Wishlist

BROOKHAVEN NATIONAL LABORATORY

- **Templates templates templates.**
  - libsigc++ is heavily templated and currently can not be interpreted by CINT. Only compiled code can enjoy libsigc++ and this code must have `#ifndef __CINT__` liberally sprinkled about.
  - More and more user code is turning to templates for solutions (eg. `Range`).
- Optional parent-owns-children widget memory management simplifies GUI writing. Remove need for creating and managing superfluous TGLayoutHints.
- Even with simplified wrappers, writing GUIs by hand is tedious. Look to *eg.* GLADE as good example of GUI builder and libglade for dynamically created GUIs. User can rearrange a libglade app's GUI via the GLADE app **with out** recompiling by generating new XML file.

ROOT in Minos MC/Sim framework design

ROOT Geometric Modeller

- As the Minos geometry representation for MC/SIM and reconstruction
- With standard tools for track swimming through a segmented detector
- With visualization capabilities

Concrete implementations of virtual Monte Carlo for G3, G4, …

We hope to see HEP progress in standardizing its particle list and PDG classes, including reconciliation of StdHep and TParticle.