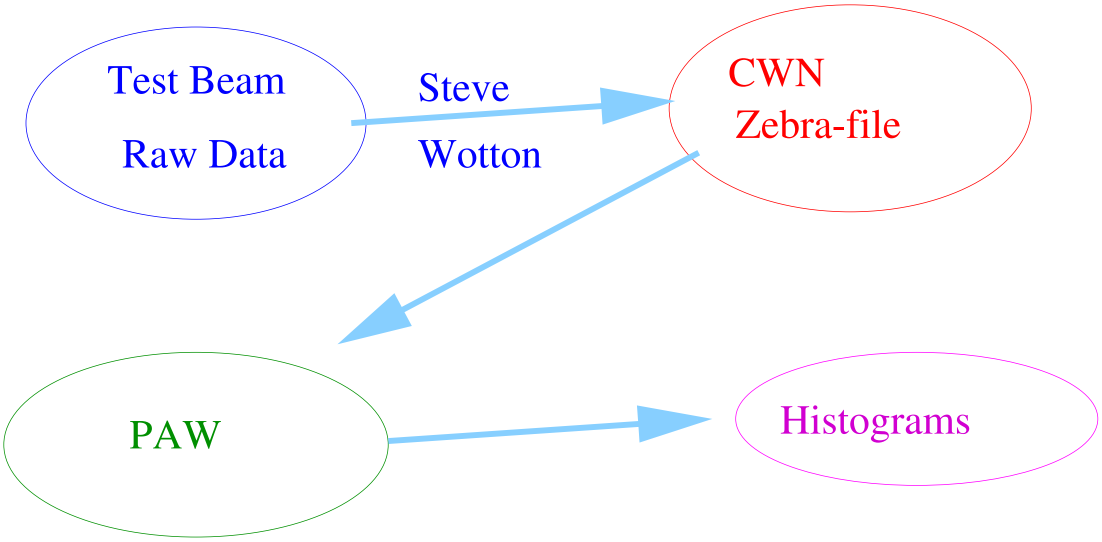
LHC++ for test-beam data

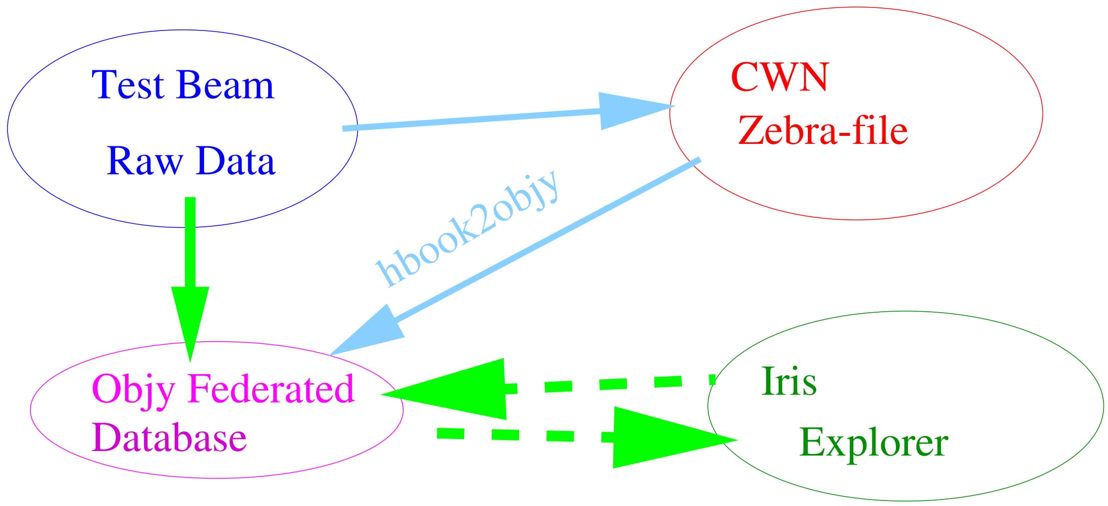
LHC++ would like to meet the requirements of LHC experiments for a "CERNLIB-like" environment.

With HBOOK:



CWN

What I did with LHC++



Hbook2objy package
(`http://wwwinfo/asd/lhc++/HepExplorer/ntkit/`)
from Dino Ferrero in order to have a skeleton.

CWN

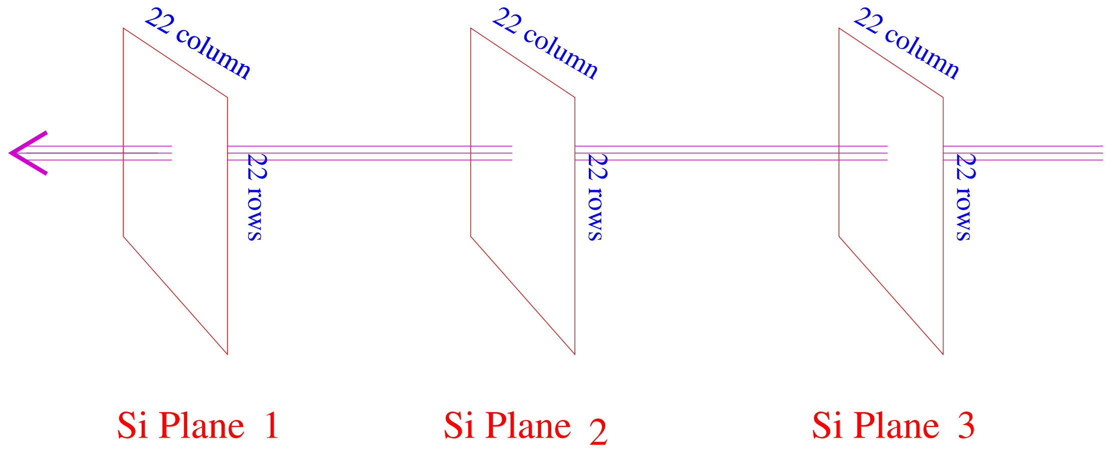
In the Test Beam, hbook libraries are used, and the out coming n-tuple has the following structure:

```
*******************************************
* Ntuple ID=10  Entries = 10240   RICH
*******************************************
* Type * Range      *  Block    *  Name     *
*******************************************
* I*4  * [0,1280] * HPD       * nch
* U*4  * [0,1791] * HPD       * id(nch)
* R*4  *          * HPD       * adc(nch)
* R*4  *          * HPD       * norm(nch)
* U*4  * [0,61]   * HPD       * pixid(nch)
* I*4  * [0,1536] * SI        * sinch
* I*4  *          * SI        * siid(sinch)
* R*4  *          * SI        * siadc(sinch)
********************************************
```

Silicon Tracker

For the time being, only the the tracker has been taken in account.

It consists of three layers, each one with 484 pixel.



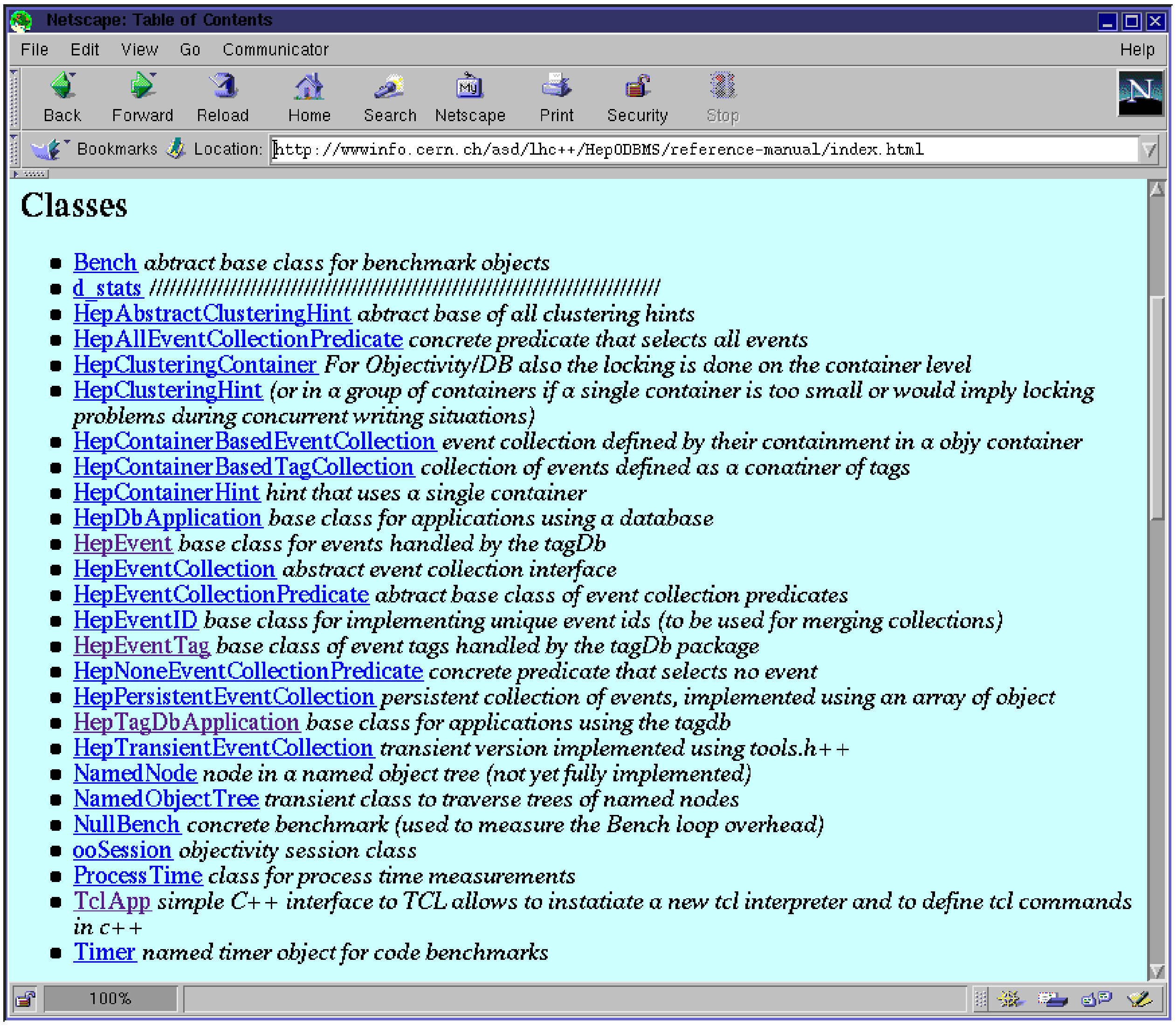
Files

The files are those:

- buildObjy.mk
- HP-UX
  - SiliconSource
  - libSilicon.sl
- SiliconSource.cpp
- Silicon.cpp
- Silicon.ddl
- Silicon.h
- adcDisplay.cpp
- adcDisplay.h
- adcDisplay.sl
- adcLoop.map
- siDisplay.cpp
- siDisplay.h

Classes used

HepODBMS provides the following classes (the User Manual is still on Beta version):



Classes

- Bench *abtract base class for benchmark objects*
- d_stats *//*
- HepAbstractClusteringHint *abtract base of all clustering hints*
- HepAllEventCollectionPredicate *concrete predicate that selects all events*
- HepClusteringContainer *For Objectivity/DB also the locking is done on the container level*
- HepClusteringHint *(or in a group of containers if a single container is too small or would imply locking problems during concurrent writing situations)*
- HepContainerBasedEventCollection *event collection defined by their containment in a objy container*
- HepContainerBasedTagCollection *collection of events defined as a conatiner of tags*
- HepContainerHint *hint that uses a single container*
- HepDbApplication *base class for applications using a database*
- HepEvent *base class for events handled by the tagDb*
- HepEventCollection *abstract event collection interface*
- HepEventCollectionPredicate *abtract base class of event collection predicates*
- HepEventID *base class for implementing unique event ids (to be used for merging collections)*
- HepEventTag *base class of event tags handled by the tagDb package*
- HepNoneEventCollectionPredicate *concrete predicate that selects no event*
- HepPersistentEventCollection *persistent collection of events, implemented using an array of object*
- HepTagDbApplication *base class for applications using the tagdb*
- HepTransientEventCollection *transient version implemented using tools.h++*
- NamedNode *node in a named object tree (not yet fully implemented)*
- NamedObjectTree *transient class to traverse trees of named nodes*
- NullBench *concrete benchmark (used to measure the Bench loop overhead)*
- ooSession *objectivity session class*
- ProcessTime *class for process time measurements*
- TclApp *simple C++ interface to TCL allows to instatiate a new tcl interpreter and to define tcl commands in c++*
- Timer *named timer object for code benchmarks*

Classes Initialisation

The Silicon.ddl file defines the classes used:

```
File  Edit  Apps  Options  Buffers  Tools  C++                                  Help
#ifndef Silicon_ddl
#define Silicon_ddl
#include "localSchema.h"
#include "HepODBMS/tagdb/HepEvent.h"
#include "HepODBMS/tagdb/HepEventTag.h"
#include "HepODBMS/odbms/HepVArray.h"


class mySilicon {
public:
d_Long siid;
d_Float siadc;
  mySilicon () {};
};

declare (HepVArray,mySilicon);

class SiliconEvent : public HepEvent {
public :
d_Long sinch;
HepVArray(mySilicon) silicon;

public:
  HepHintDeclare;
  SiliconEvent () {};
};

class SiliconTag : public HepEventTag {
public:
d_Long sinch;
public:
  // Both "tagUpdate" et "checkDescr" are implemented in
  // Silicon.cpp
  int tagUpdate();
  static  int  checkDescr(char *);
  HepRef(SiliconEvent) getEvent(){return (HepRef(SiliconEvent)&)event();}
  HepHintDeclare;
};

#endif
-----XEmacs: Silicon.ddl        (C++ Font)----28/19--Top---------------------
No further undo information
```

The Application Source code

The Application Source code has been heavily changed, inserting and adapting Steve's source for reading the raw file.

The main program is indeed very short:

```
int main() {
  fillDb myApp("SiliconTag");
  myApp.execute("SiliconTag");
}
```

And here is the function for filling the database:

File Edit Apps Options Buffers Tools C++ Help

```
int readEvent()
{
  HepRef(SiliconEvent) theEvt;
  HepRef(SiliconTag) tag;
  int i;
  theEvt = new (SiliconEvent::clustering()) SiliconEvent;
  tag = new (SiliconTag::clustering()) SiliconTag;

  // Fill the event
  // Scalars first
   theEvt->sinch = si.nch;

  // Fixed arrays
  // Variable arrays
   theEvt->silicon.resize((unsigned int)si.nch);
   for (i=0;i<si.nch;i++) {
     theEvt->silicon[i].siid = si.id[i];
     theEvt->silicon[i].siadc = si.adc[i];
   }
  // Replicate data in the Tag
  tag->setEvent(theEvt);
  tag->tagUpdate();
  return 1;
```

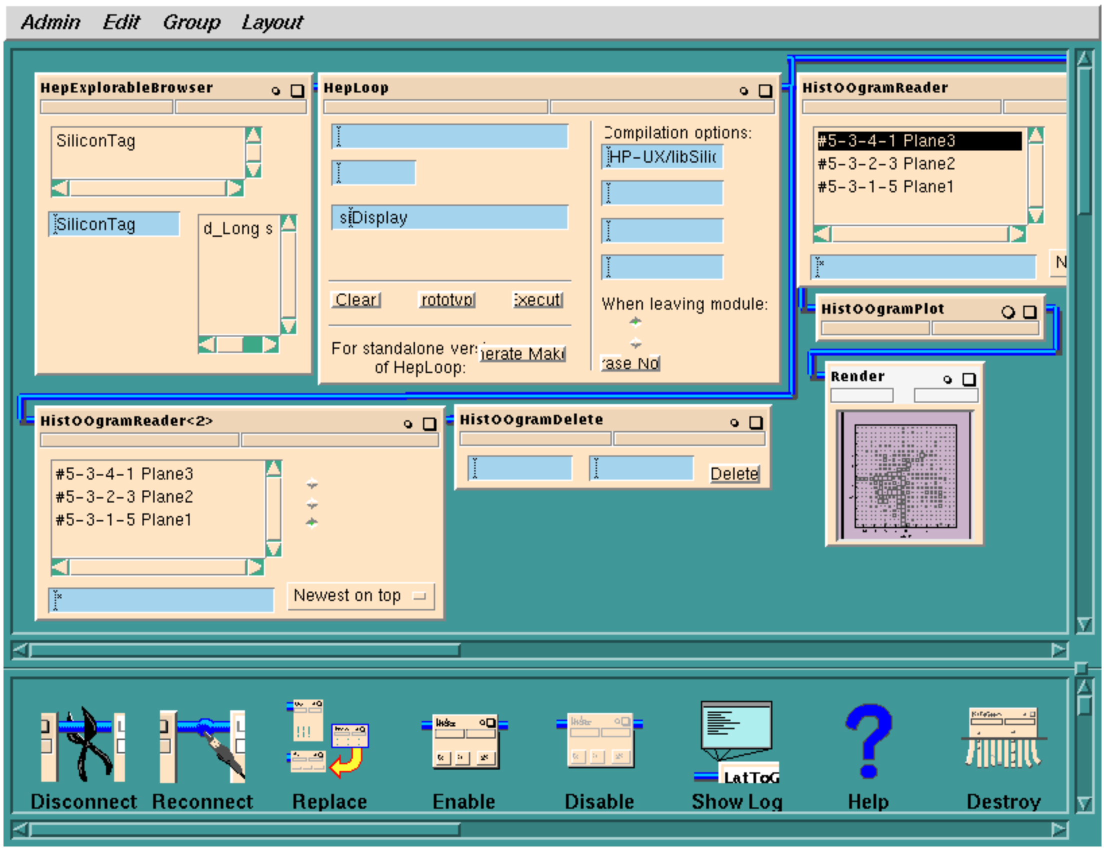
```
--**- tmp    <V>     (C++ Font)----All-------------------------
Font -*-Courier-Medium-R-*-*-*-180-75-75-*-*-ISO8859-1
```

Reading the DataBase

The DB has now the following structure (a bit more structured than in the n-tuple):

The event is an array of variable length of type "`mySilicon`", where "`mySilicon`" is a class with two methods - `siid` and `siadc`.

For reading it we use Iris Explorer:



Looping the DB

In HBOOK-language what I do with Explorer:

- Create histograms
- Write a program calling HBOOK libraries to loop the n-tuples
- Fill the histograms

Because of the variability of the array-size we need to read the size every time and do a loop with the index equal to the size.

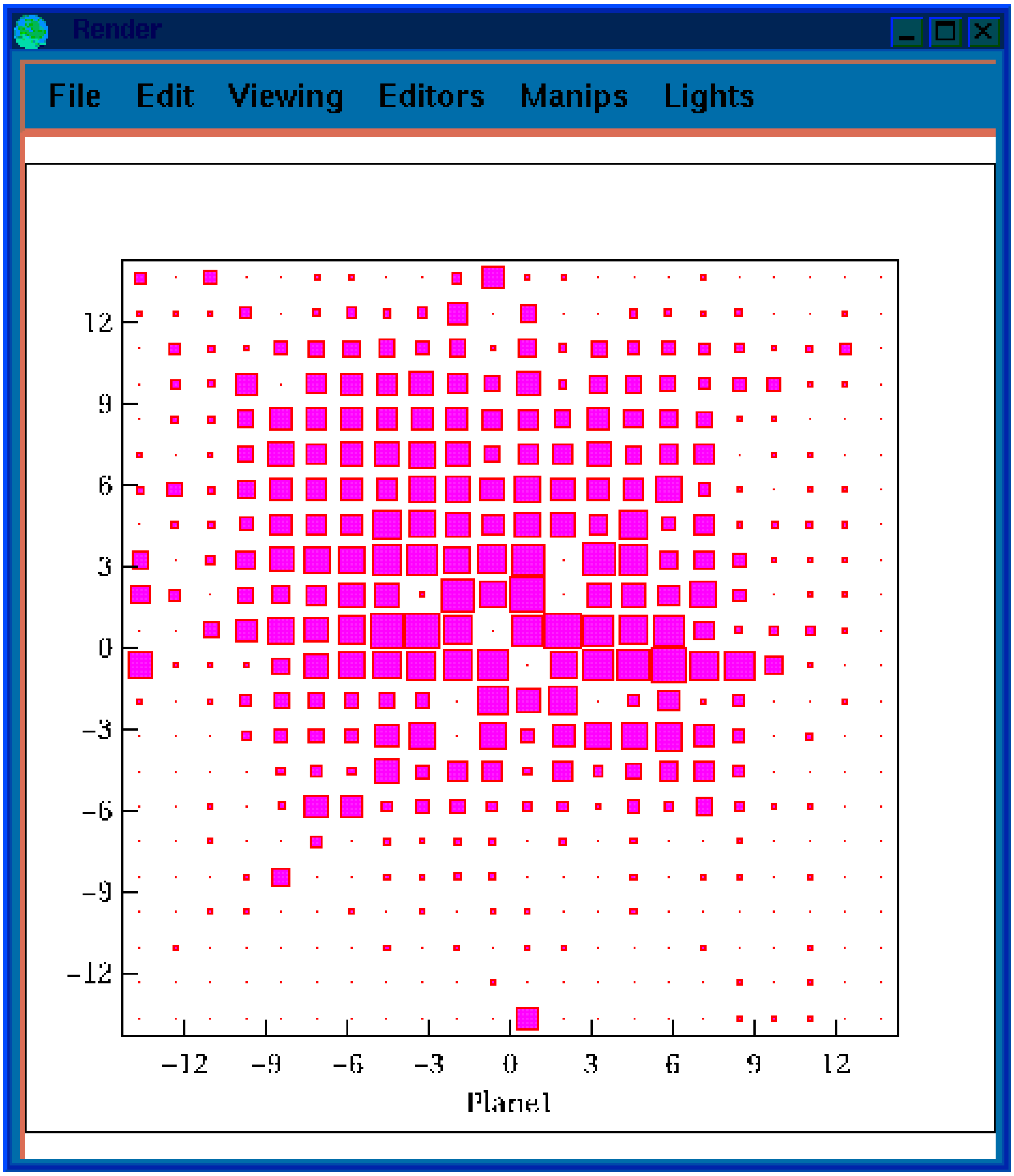
Looping the DB

The HepLoop module creates a Prototype code and a `GNUmakefile`. We have to complete it:

```
Buffers Files Tools Edit Search Mule C++ Help
// adcDisplay.cpp

//  This file has been automatically generated.
//....

int GeneratedInterHisto::match(){

        // Define your cut/match expression here.
        // ***** BEGIN USER CODE: *****
  return 1;
        // *****   END USER CODE   *****}

void GeneratedInterHisto::action(){
        // Action to be executed when `match' succeeds.
        // ***** BEGIN USER CODE: *****

  HepRef(SiliconTag) theTag;
  getCollection()->getCurrentObject(theTag);
  for (unsigned int i=0; i<sinch; i++) {
    unsigned int id = theTag->getEvent()->silicon[i].siid;
    siId  si_id(id);
    if(si_id.plane()==1)
      Plane1->fill(si_id.x(),si_id.y(),1.);
    else if (si_id.plane()==2)
      Plane2->fill(si_id.x(),si_id.y(),1.);
    else if (si_id.plane()==3)
      Plane3->fill(si_id.x(),si_id.y(),1.);  }

        // *****   END USER CODE   *****
}

int GeneratedInterHisto::preExecute()
{
        // System init:
        // Find the explorable collection, which can be
        // accessed later on via `getCollection()'.
        //
        if( ! InterNTLoop::preExecute() ) return 0;

        // ***** BEGIN USER CODE: *****
        Plane1 = NEW_HISTO Histo2D("Plane1", 22,-14.95,14.95,22,-14.95,14.95);
        Plane2 = NEW_HISTO Histo2D("Plane2", 22,-14.95,14.95,22,-14.95,14.95);
        Plane3 = NEW_HISTO Histo2D("Plane3", 22,-14.95,14.95,22,-14.95,14.95);
        // *****   END USER CODE   *****

        return 1; // Operation has been successful.
}

--:--  tmp              (C++)--L2--All------------------------------------
Wrote /afs/cern.ch/user/m/malemib/HP-UX/adcLoop/tmp
```

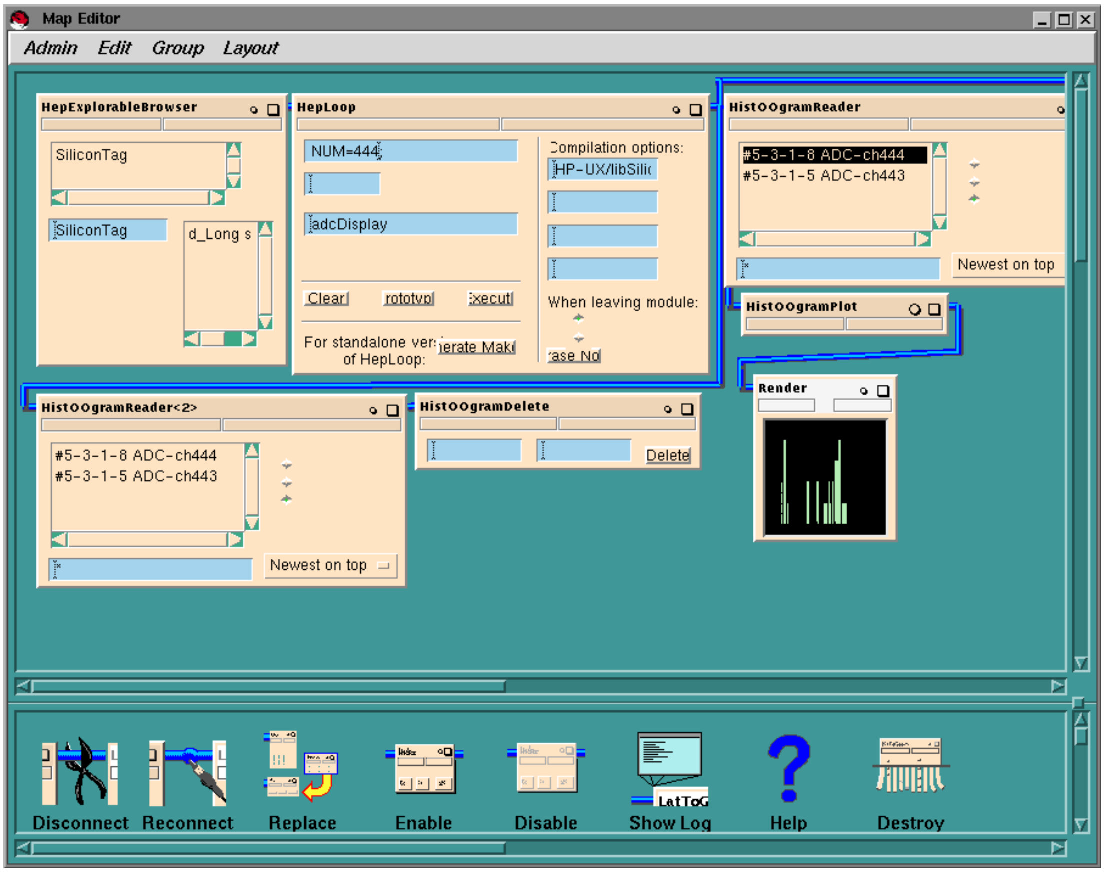
Histograms!

Finally we can compile and run it, we get the histograms and we can read it through the HistogramReader ⇒ Render Modules



ADC spectra

It is possible to set variables in order to create histograms without compile every time.



Observations

"The idea is to speed up queries by defining for each **event** a Tag, i.e. a small collection of **its** most important physics attributes plus an association with the event where the Tag data come from".

The Tag model has been chosen for BIG amount of data. All the data are on-line. Often used data are once more reclustered using experiment-wide concrete tags), but all data remain inside the same (federated) database.

Documentation is still not complete and the software is ending through stable releases.

Notes

- It works
- You need OpenGL libraries on your display.
  - Xterminal: work on hpplus;
  - Linux-PC: Objy already available, Iris Explorer announced release in 1999: in the meantime work on hpplus
  - WindowsNT-PC: OK locally (sic dicitur), CERN is going to acquire GL extension for Exceed.
- The Objy AMS (Advanced Multi-threaded Server) is not installed. You better use always the same machine (otherwise run `ooinstallfd`).

- Once you run IrisExplorer on a Federated Database you can not swap to another Federated Database.
- Every thing is compiled, and no interpreter will come.
- The HepExplorableBrowser Module of IrisExplorer is not so powerful like the `ootoolmgr`, who can browse through the Tag and Event.