5

Histograms And N-tuples

Gaudi Framework Tutorial, 2001

Schedule:	**Timing**	**Topic**
	15 minutes	Lecture
	20 minutes	Practice
	35 minutes	Total

Objectives

After completing this lesson, you should be able to:

- **Book and fill histograms.**
- **Book and fill N-tuples.**

Lesson Aim

The aim of this topic is to introduce histograms and N-tuples as they were introduced in Gaudi. It will be shown how to book histograms and N-tuples and how to fill them.

Histograms & N-tuples

- One of they key tools in HEP
- HBOOK was one of the best packages in CERNLIB
- Usage and function is obvious
  - We did not reinvent the wheel
- In Gaudi it's the same concept
  - First book then Fill

Histograms & N-tuples

Histograms and N-tuples are one of the key concepts for data analysis in high energy physics. In fact, HBOOK was one of the most successful packages in CERNLIB, both for batch processing and interactive use within PAW.
The usage of HBOOK was obvious. Within Gaudi the wheel was not re-invented, but is rather based on the known concepts:

•First the histogram or the N-tuple must be booked

•Then it can be filled.

Histograms - Good To Know...

- Histograms are kept in memory
- If not saved - they are lost
- Like all other data - they reside in a Data Store
  - Same access mechanism
- Persistency is configuarble
  - HBOOK, ROOT

Histograms - Good To Know

Like in the old CERNLIB approach histograms are kept in memory. This allows fast access for filling and other operations without unnecessary I/O.

However, this means that the histograms must be saved at the end of the job. Otherwise the histograms are lost as soon as the process terminates.

Histograms within Gaudi reside on a data store - similar to event data. This allows to use the same access mechanism.

The persistency mechanism for histograms in Gaudi is configurable. Persistent back-ends exist using HBOOK or ROOT. Which one you prefer depends on your preference for the interactive analysis.

Booking 1-d Histograms

- **Ask the Histogram service**

```
IHistogram1D* multiplicityH1D =
  histoSvc()->book("/stat/simple/1",
                   "Visible charged Multiplicty",
                   100,
                   0,
                   1000.0);
```

Title	"Visible charged Multiplicty",
Number of bins	100,
Low Edge	0,
High Edge	1000.0);

- **“/stat/simple/1”**
  - /stat: **Opt.**
  - /simple: **RZ-Directory**

Theory: Histogram identifier (short name)
Practice: HBOOK histogram ID

Booking 1-d Histograms

Histograms are delivered by the histogram service. Any client can ask this service to create a histogram. Delivered to the user is a pointer to the created histogram, which then can be used to invoke the necessary actions like e.g. filling.

A histogram - like in HBOOK has an identifier, which represents the location on the data store. The other arguments:

•the histogram title

•the number of bins

•the lower edge of the histogram and

•the upper edge are the same as in HBOOK.

The Identifier of the histogram has three parts. The first one is fixed, it is the entry point to the transient store. The second represents a directory structure, which in case of HBOOK will be reflected in the resulting RZ file (HBOOK does not allow directory names to be longer than 8 characters!). The third part is the identifier of this histogram object itself. In principle this can be any string. In practice, when using HBOOK as a persistent back-end this must be a number.

Booking 2-d Histograms

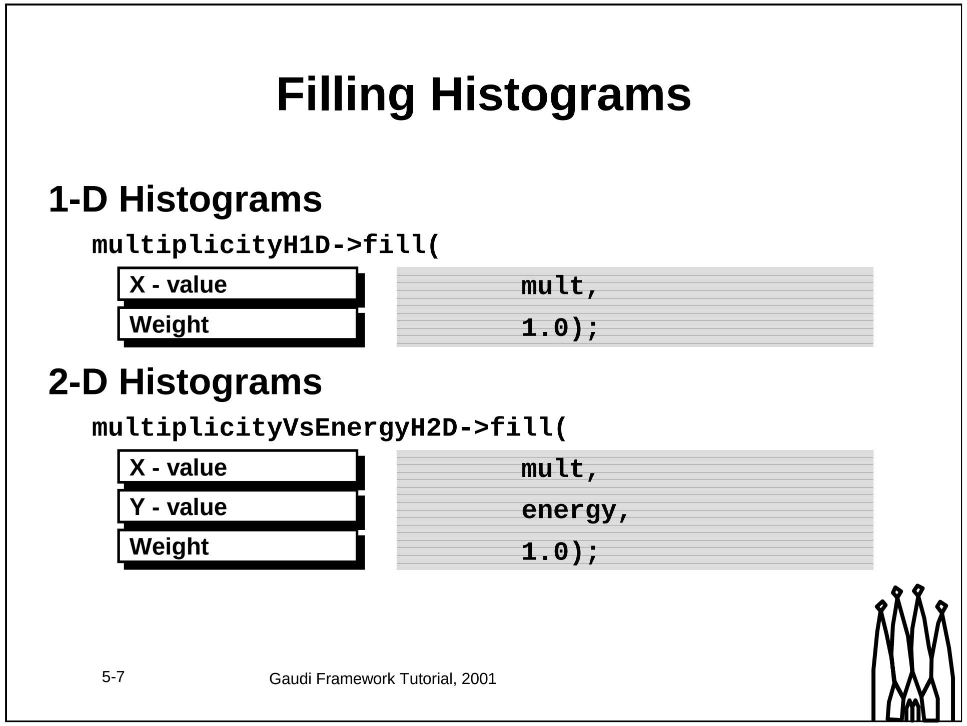
Ask the Histogram service

```
IHistogram2D* multiplicityVsEnergyH2D =
  histoSvc()->book("simple/2",
                   "Multiplicty vs. Energy (GeV)",
                   100,
                   0.0,
                   1000.0,
                   50,
                   0.0,
                   500.0);
```

Title	"Multiplicty vs. Energy (GeV)",
Number of bins (X)	100,
Low Edge (X)	0.0,
High Edge (X)	1000.0,
Number of bins (Y)	50,
Low Edge (Y)	0.0,
High Edge (Y)	500.0);

Booking 2-d Histograms

There is not a lot to be explained. It’s like the 1-d histograms. In addition the number of bins, low and high edge for the second dimension must be specified.



Filling Histograms

1-D Histograms

```
multiplicityH1D->fill(
```

X - value	`mult,`
Weight	`1.0);`

2-D Histograms

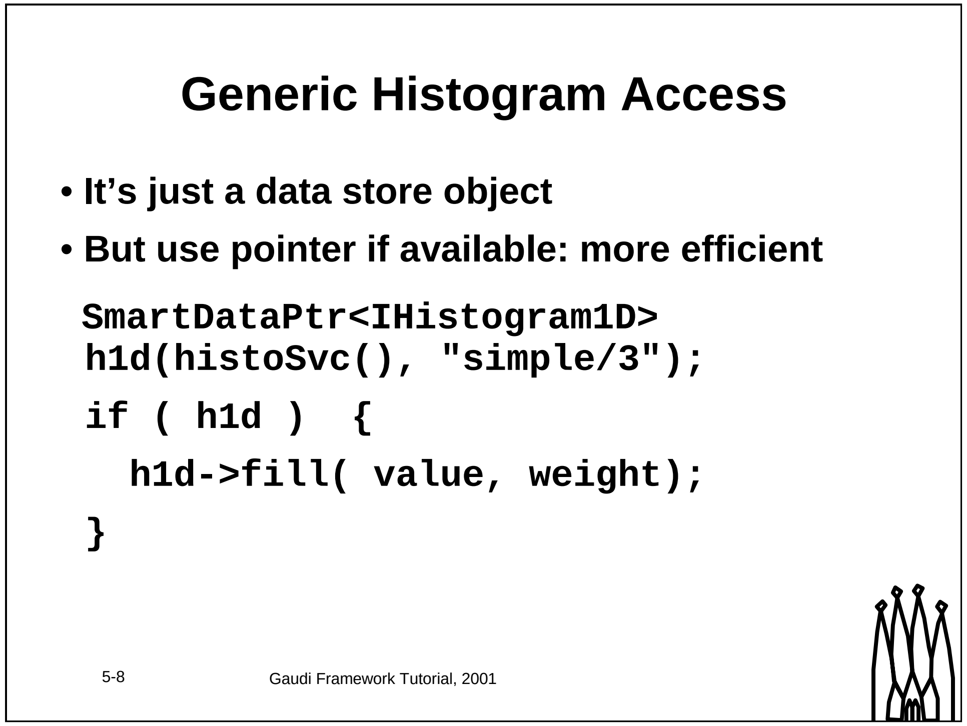
```
multiplicityVsEnergyH2D->fill(
```

X - value	`mult,`
Y - value	`energy,`
Weight	`1.0);`

5-7 Gaudi Framework Tutorial, 2001

Filling Histograms

Filling 1-d or 2-d histograms is similar to the HBOOK calls HF1 and HF2. The pointer to the histogram already specifies the histogram. Hence it is not necessary to specify as an extra argument the histogram identifier.



Generic Histogram Access

- It's just a data store object
- But use pointer if available: more efficient

```
SmartDataPtr<IHistogram1D>
h1d(histoSvc(), "simple/3");
if ( h1d )  {
  h1d->fill( value, weight);
}
```

5-8 Gaudi Framework Tutorial, 2001

Generic Histogram Access

Histograms are data store objects like any other. The example code shows that the histogram can be accessed like any other store object.

Note:

The usage shown above is less introduces additional overhead compared to the usage of the pointer directly. However, this usage sometimes is necessary e.g. if the histogram was booked in one algorithm, but has to be manipulated in another algorithm. However, this should only be used for short term tests, because it also introduced run-time coupling.

Histogram Persistency

Job options

```
// Standard HBOOK setup
#include “$STDOPTS/Hbook.opts”;
// Output file (for HBOOK, ROOT)

HistogramPersistencySvc.OutputFile = "histo.hbook";
```

5-9 Gaudi Framework Tutorial, 2001

Histogram Persistency

The persistent back-end for the histograms is defined in the job options file.

First the persistency type must be specified. Currently histograms can be saved using HBOOK or ROOT. The type NONE means that the histograms should not be made persistent.

If histogram persistency is desired the corresponding output file must be specified.

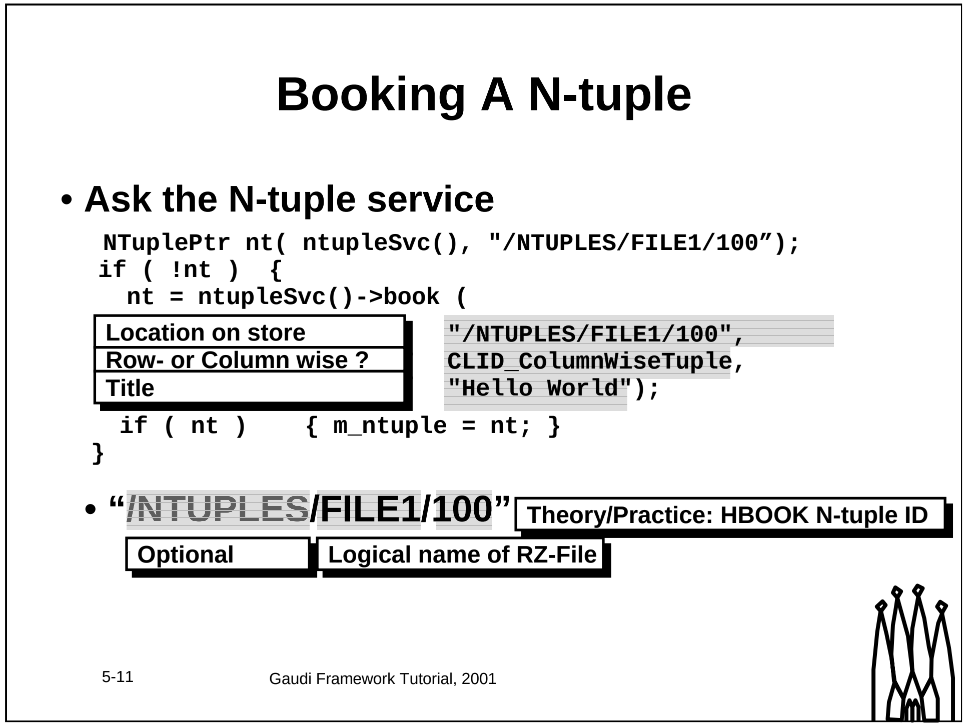
N-tuples - Good To Know...

- **Cannot be kept in memory**
  - **Grow and grow and grow...**
- **Like all other data - reside in a Data Store**
  - **Same access mechanism**

5-10 Gaudi Framework Tutorial, 2001

N-tuples - Good to know

N-tuples - unlike histograms - cannot be kept in memory. Because there are constantly rows added the size of the N-tuple is not constant. They grow and grow. N-tuples must be connected to a persistent back-end already when filled.
Like all other data, N-tuples are also data store objects.



Booking A N-tuple

- **Ask the N-tuple service**

```
 NTuplePtr nt( ntupleSvc(), "/NTUPLES/FILE1/100”);
 if ( !nt )  {
   nt = ntupleSvc()->book (
                          "/NTUPLES/FILE1/100",
                          CLID_ColumnWiseTuple,
                          "Hello World");
   if ( nt )    { m_ntuple = nt; }
 }
```

Location on store
Row- or Column wise ?
Title

- **“/NTUPLES/FILE1/100”**

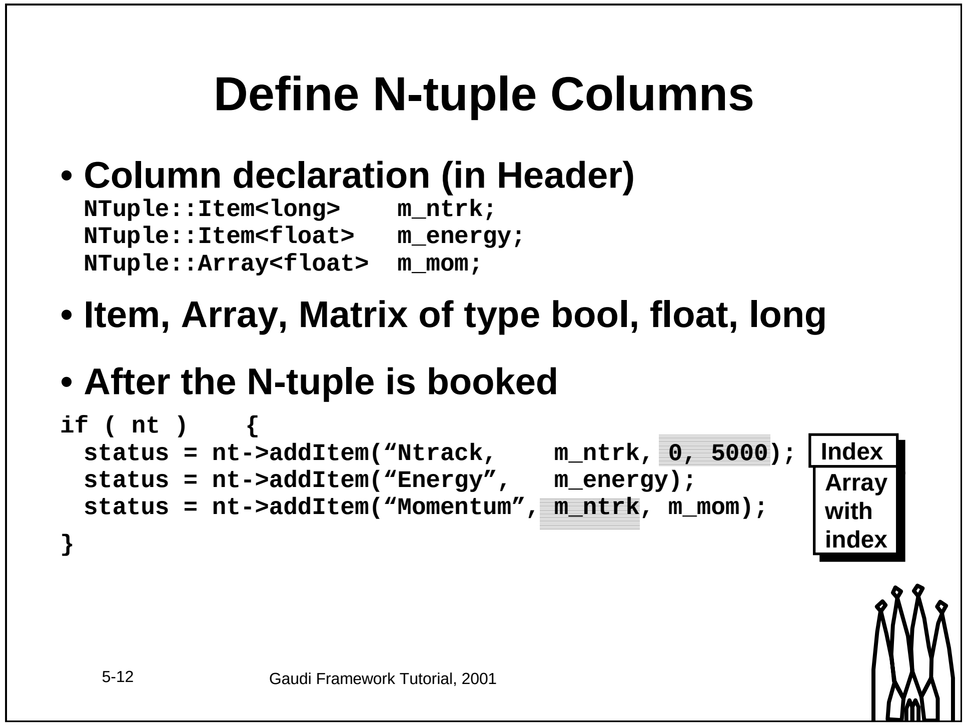
Theory/Practice: HBOOK N-tuple ID
Optional
Logical name of RZ-File

5-11 Gaudi Framework Tutorial, 2001

Booking an N-tuple

N-tuples can be booked using the N-tuple service. When being booked the N-tuple identifier, the type (Row-wise or Column-wise) and the title must be specified.

However, for N-tuples the identifier is a slightly different meaning. The first sub-specifier is the logical name of the N-tuple file. N-tuples can be saved to different files if desired. The N-tuple identifier itself must obey the same constraints as histograms. If the N-tuple is going to be saved using HBOOK, it must be a number.



Define N-tuple Columns

- **Column declaration (in Header)**

```
NTuple::Item<long>    m_ntrk;
NTuple::Item<float>   m_energy;
NTuple::Array<float>  m_mom;
```

- **Item, Array, Matrix of type bool, float, long**
- **After the N-tuple is booked**

```
if ( nt )    {
  status = nt->addItem(“Ntrack,    m_ntrk, 0, 5000);
  status = nt->addItem(“Energy”,   m_energy);
  status = nt->addItem(“Momentum”, m_ntrk, m_mom);
}
```

Index
Array with index

5-12 Gaudi Framework Tutorial, 2001

Define N-tuple Columns

Although the N-tuple is now booked, it is not yet really usable. The N-tuple columns need specification.

Row-wise N-tuples only accept individual numbers (Ntuple::Item<…>). Usable types are bool, float and long/unsigned long).

Column-wise N-tuples are more general and dynamic. Gaudi accepts individual numbers, arrays and matrices. However, N-tuples may only have one index column (a la HBOOK).

The above code fragments would correspond to the following FORTRAN code:

```
PARAMETER (MAXTRK = 5000)
INTEGER   NTRACK
REAL*4    ENERGY
REAL*4    MOMENTUM(MAXTRK)
COMMON /CMTRK/ NTRACK, ENERGY, MOMENTUM
CALL HBNAME(ID,'VARBLOK2',
           NTRACK,
          'NTRACK[0,5000]:I, '//
          'ENERGY:R, ’//
          'MOMENTUM(NTRACK):R')
```

Note:
Do not use double types for N-tuples. At least PAW gets highly confused. In addition for HBOOK the total number of double columns is highly limited.

Filling N-tuples

- **Item behaves like a number**
  - Just assign values: m_energy = 50.67;
- **Array like an array**
  - m_mom[i] = 10.25;
- **Write record**

```
status = m_ntuple->write();
if ( !status.isSuccess() )  {
  log << MSG::ERROR << "Cannot fill N-tuple" << endreq;
}
```

Filling N-tuples

The N-tuple item behaves like a number. You can just assign a value, which later will be stored in the N-tuple.

Similar is the treatment of arrays and matrices.

Once all items are assigned to the N-tuple the tuple can be written to disk. This is done by telling the N-tuple service to write a record of the specified N-tuple.

Note:

For performance reasons heavy arithmetic operations should not be performed on the items of the N-tuple.

N-tuple Persistency

- **Job options**

```
NTupleSvc.Output = { ”FILE1
                     DATAFILE=’../job/tuples.hbook'
                     OPT='NEW'
                     TYP=’HBOOK'" };
```

Technology: HBOOK

```
NTupleSvc.Output = { ”FILE1
                     DATAFILE=’../job/tuples.root'
                     OPT='NEW'
                     TYP='ROOT'" };
```

ROOT

N-tuple Persistency

Output-files for N-tuples are specified in the job-options, similar to the specification of event data input. The first tag is the logical name of the file.

Note:

•“FILEn” is the logical name you can refer to the file within Gaudi. You could use any other name as well like “ECAL” or “Frank”.

•Full support of column wise N-tuples for interactive usage is only given for HBOOK. Otherwise column wise N-tuples are written as binary large objects (BLOB), which can only be read back using Gaudi itself.

Hands On

Files: VisibleEnergyAlgorithm.h/cpp

Include additional header files in cpp

```
#include "GaudiKernel/MsgStream.h"
#include "GaudiKernel/SmartDataPtr.h"
#include "CLHEP/Units/PhysicalConstants.h"
#include "LHCbEvent/AxPartCandidate.h"
#include "GaudiKernel/IHistogramSvc.h"
#include "GaudiKernel/IHistogram1D.h"
#include "LHCbEvent/Event.h"
#include "GaudiKernel/INTupleSvc.h"
```

5-15 Gaudi Framework Tutorial, 2001

Hands On: Histograms

Edit VisibleEnergyAlgorithm.h

- **Add foreward declaration**
  ```
  class IHistogram1D;
  ```
- **Add Pointers**
  ```
  IHistogram1D* m_h1dMom;
  IHistogram1D* m_h1dMult;
  IHistogram1D* m_h1dMomRatio;
  ```
- **Initialize pointers in the implementation!**

5-16 Gaudi Framework Tutorial, 2001

Hands On: Book Histogram

- **Whatever you want to histogram**
- **…or:**

```
histoSvc()->book("simple/1",
                 "Visible charged Multiplicty",
                 100, 0, 1000.0);
histoSvc()->book("simple/2",
                 "Visible charged Mult. - Ratio",
                 100, 0, 1.0);
histoSvc()->book("simple/3",
                 ”Momentum of AX part.candidates",
                 100, 0, 250.0);
```

Hands On: Fill Histogram

```
SmartDataPtr<AxPartCandidateVector>
cands (eventSvc(), "/Event/Anal/AxPartCandidates");
if ( ! cands ) {
  … HANDLE ERROR …
}
// Loop over tracks and fill simple histogram
AxPartCandidateVector::const_iterator i;
for(i = cands->begin(); i != cands->end(); i++) {
  m_h1dMom->fill((*i)->fitMomentum()/GeV, 1.0);
}
```

**Geant 4 units are MeV, mm, nanosec.
Need to convert to something reasonable**

Hands On: Add N-tuple

- **Edit VisibleEnergyAlgorithm.h**
  - **Add** `#include “GaudiKernel/NTuple.h”`
  - **Add N-tuple columns**

```
NTuple::Item<long>              m_ntrk;
NTuple::Item<float>             m_energy;
...
```

Hands On: Book N-tuple

```
 NTuplePtr nt(ntupleSvc(), "FILE1/Collection");
 if( !nt )  {
   nt = ntupleSvc()->book("FILE1/Collection",
                          CLID_RowWiseTuple,
                          "Hello World");
  if ( nt )    {
    status = nt->addItem ("Ntrack",   m_ntrk);
    status = nt->addItem (”Energy",   m_energy);
    m_ntuple = nt;
  }
  else    {   // did not manage to book the N tuple....
    return StatusCode::FAILURE;
  }
}
```

Hands On: Fill N-tuple

```
// Assign values
m_ntrk    = <number of tracks>;
m_energy  = <energy>;

// Write record
status = m_ntuple->write();
if ( !status.isSuccess() )   {
  log << MSG::ERROR << "Cannot fill NTuple" << endreq;
}
// After writing values are zeroed.
```

Visualize Histograms & N-tuple

…using PAW

```
PAW> hi/file 1 histo.hbook
PAW> cd simple
PAW> hi/pl 1
PAW> hi/pl 2
PAW> hi/pl 3
PAW> hi/file 2 ntuple.hbook
PAW> nt/pri 100
PAW> nt/pl 100.Ntrack
PAW> nt/pl 100.Momentum
PAW> quit
```