Analysis and data persistence in Geant4

`Geant4-11.1` reference - Based on previous Geant4 courses

Lorenzo Pezzotti
CERN EP-SFT

Code for this tutorial on `GitHub`

10th International Geant4 Course in Korea 2023
@ Jeju National University

Introduction

The default `Geant4` simulation flow sets up the geometry, the physics list and the primary generator, transports particles and… *that's it, no data permanently stored!*

Bash - exampleB1 execution

```
$ source /path-to/geant4_11.1.2-install/bin/geant4.sh
$ cmake -DGeant4_DIR=/path-to/geant4_11.1.2-install/lib/Geant4-11.1.2/ /path-to/
geant4-11.1.2/examples/basic/B1/ .
$ make
$ ./exampleB1 run1.mac

**************************************************************
 Geant4 version Name: geant4-11-01-patch-02 [MT]
  << in Multi-threaded mode >>

$ ls

CMakeCache.txt  Makefile   exampleB1  exampleB1.out  run1.mac  vis.mac   CMakeFiles
cmake_install.cmake   exampleB1.in   init_vis.mac   run2.mac
```

… no data

g4analysis (1/2)

- In Geant4 the user is responsible for storing the variables of interest

- The Geant4 analysis category g4analysis (geant4/source/analysis/) provides a unique interface:
  - to write histograms and n-tuples (the so-called *primitive types*), and
  - to specify the format (ROOT, XML, CSV, HDF5)

- Good choice to support the large and heterogeneous user community that adopts different tools (Python, ROOT in HEP, …) to run the analysis *a posteriori*

`g4analysis` (2/2)

- `g4analysis` is available in Geant4 since December 2011
  - It is an active area of development with new features added at every release
  - It ensures input/output `(I/O)` thread-safe capability in multi-threaded simulations
    (take a second to appreciate how cool is that…)

- **NOTE:** Users can still link their applications against external libraries to handle `I/O` (typically `ROOT::TTree` for small applications and Event Data Models for large applications)

- Users access the `g4analysis` tools via the `G4AnalysisManager`

G4AnalysisManager

The `G4AnalysisManager`

- is a `singleton` (users must access it with `static` method `::Instance()`)
- handles output file(s) creation
- owns and handles histos and n-tuples

It provides

- uniform user interface to different formats (`ROOT, XML, CSV, HDF5`), *i.e.* the user is not required to know how to handle any of them,
- with high-level memory management and access to low-level objects (*e.g.* `n-tuples` columns)

and is fully integrated in the Geant4 framework (*e.g.* it is accessible via user commands and uses G4-units)

NOTE: using `hdf5` format requires `HDF5` libraries installation and `Geant4` build with

```
-DGEANT4_USE_HDF5=ON
```

Using the `G4AnalysisManager` (1/3)

The `G4AnalysisManager` usage consists of three steps

1. Create the manager, book n-tuples/histos and open a file, to be done at the begin of each run (`RunAction::BeginOfRunAction()`)
   NOTE: since `Geant4-11.0` the file format is chosen via the extension (`.root, .xml, .csv, .hdf5`)

RunAction.cc

```
#include "G4AnalysisManager.hh"

void RunAction::BeginOfRunAction(const G4Run* run) {
    auto analysisManager = G4AnalysisManager::Instance(); //get the manager

    std::string runnumber = std::to_string( run->GetRunID() );
    G4String fileName = "Run" + runnumber + ".root";  //select root format
    //G4String fileName = "Run" + runnumber + ".xml"; //or select xml format
    //G4String fileName = "Run" + runnumber + ".csv"; //or select csv format
    //G4String fileName = "Run" + runnumber + ".hdf5";//or select hdf5 format -> requires hdf5 installation
    analysisManager->OpenFile(fileName);
    //Create histogram(s)
    //
    analysisManager->CreateH1("Edep","Energy deposit",100,0.*MeV,10*GeV); //h1 - energy deposited
    // Create ntuple(s)
    //
    analysisManager->CreateNtuple("Ntuple", "Ntuple");
    analysisManager->CreateNtupleDColumn("Energy");
    analysisManager->FinishNtuple();
}
```

Using the `G4AnalysisManager` (2/3)

The `G4AnalysisManager` usage consists of three steps

2. Fill values in histos and n-tuples, likely at the end of each event (`EventAction::EndOfEventAction()`)

EventAction.cc

```
#include “G4AnalysisManager.hh"

void EventAction::EndOfEventAction(const G4Event*){

    auto analysisManager = G4AnalysisManager::Instance();

    //Fill histograms
    //
    analysisManager->FillH1(0, Edep); //Fill value as MeV
    //Fill ntuples
    //
    analysisManager->FillNtupleDColumn(0, Edep);
    analysisManager->AddNtupleRow();
}
```

Using the `G4AnalysisManager` (3/3)

The `G4AnalysisManager` usage consists of three steps

3. Write and close file, at the end of each run (`RunAction::EndOfRunAction()`)

RunAction.cc

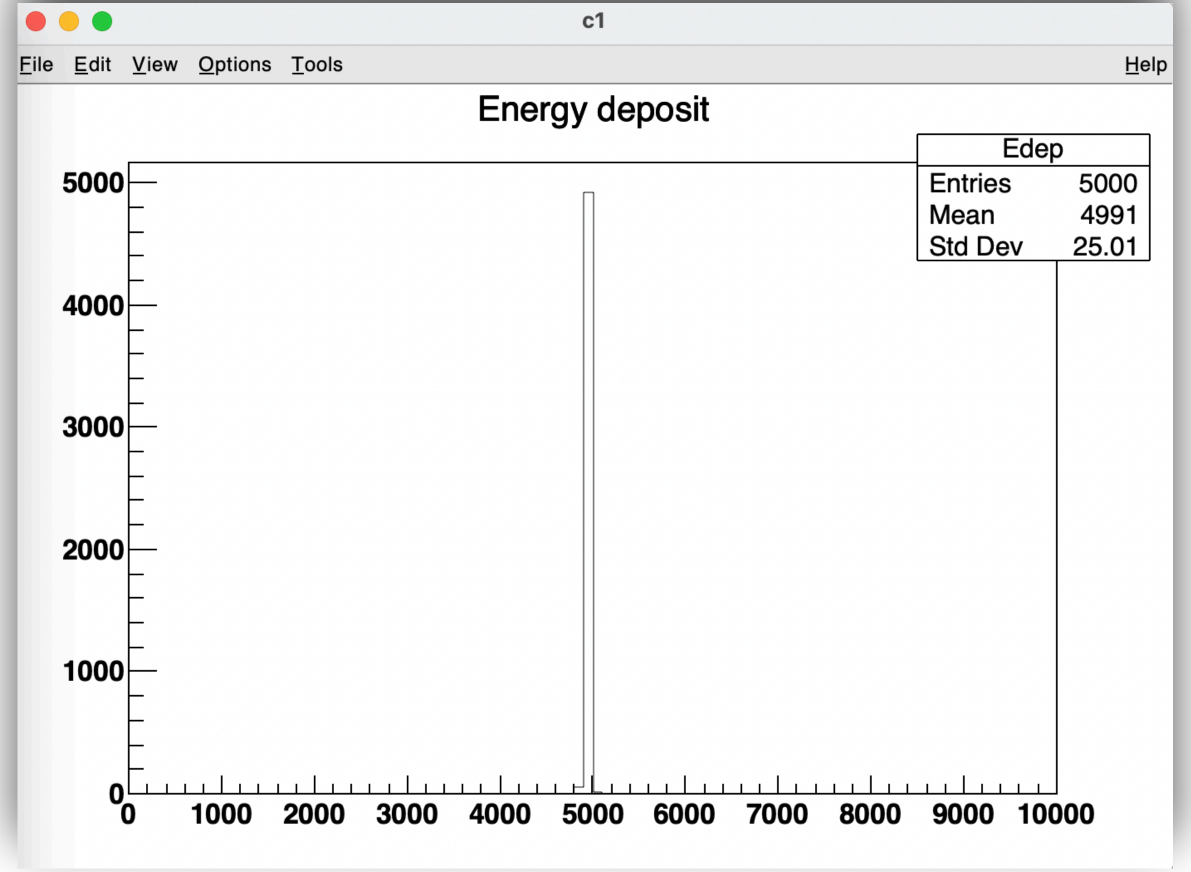
```
#include "G4AnalysisManager.hh"

void RunAction::EndOfRunAction(const G4Run*) {

    auto analysisManager = G4AnalysisManager::Instance();

    //Write and close file
    //
    analysisManager->Write();
    analysisManager->CloseFile();

}
```

NOTE: performing the steps in the suggested classes/methods is not mandatory but guarantees correct execution in multi-threaded applications

Output example - ROOT

Bash - after execution

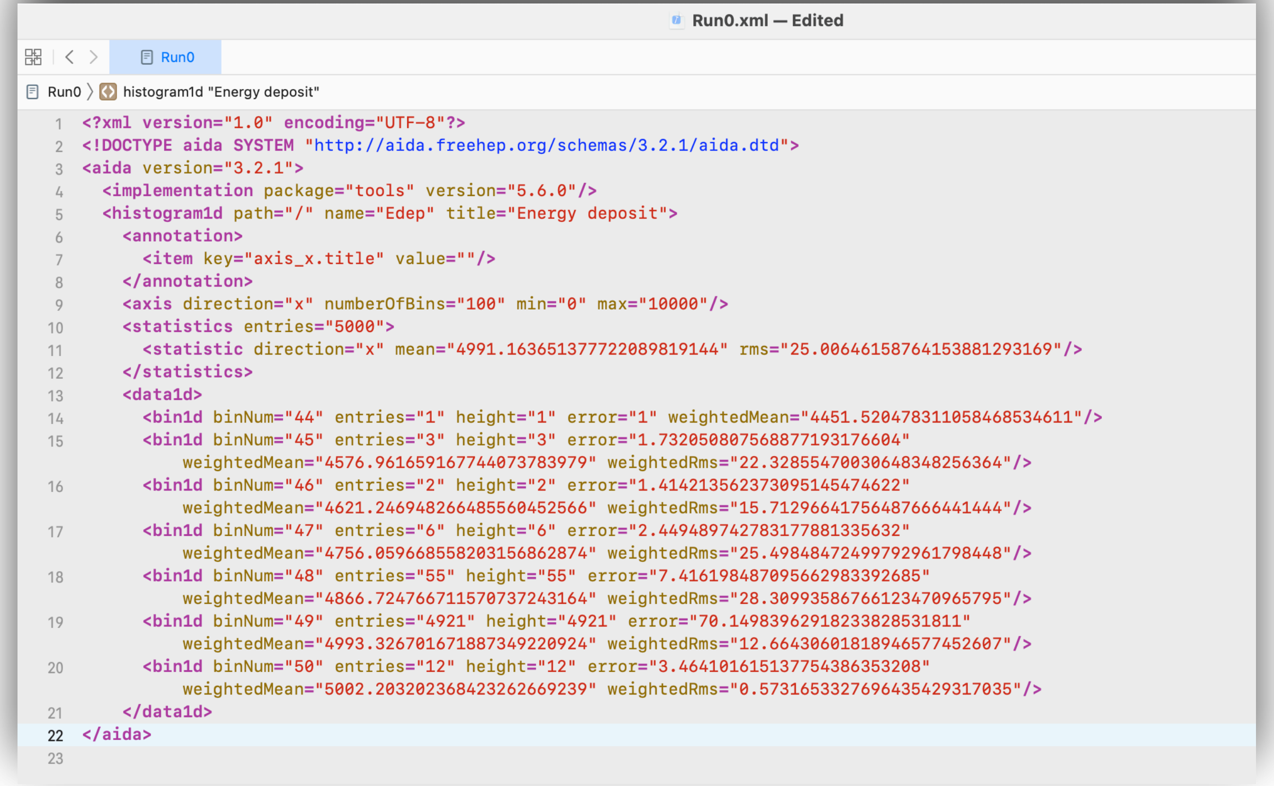
```
$ root -l Run0.root
root [0]
Attaching file Run0.root as _file0...
(TFile *) 0x1258cc990
root [1] .ls
TFile**         Run0.root
 TFile*         Run0.root
  KEY: TTree    Ntuple;1  Ntuple
  KEY: TH1D     Edep;1    Energy deposit
root [2] Edep->Draw(“histo")
root [3] Ntuple->Scan()

************************
*    Row   * Energy.En *
************************
*        0 * 4986.2465
*        1 * 4988.8028
*        2 *      5000
*        3 * 4996.4155
```



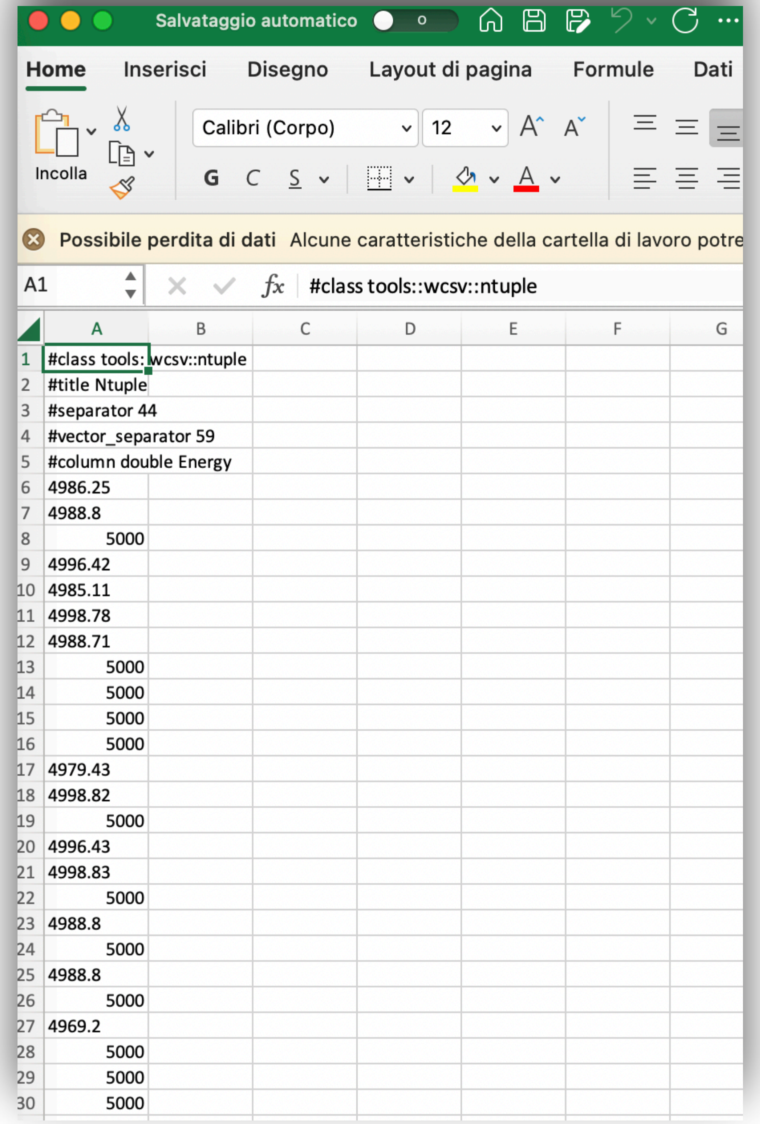
Output example - AIDA XML & CSV

Bash - after execution

```
$ open Run0_nt_Ntuple.csv
$ open Run0.xml
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE aida SYSTEM "http://aida.freehep.org/schemas/3.2.1/aida.dtd">
<aida version="3.2.1">
  <implementation package="tools" version="5.6.0"/>
  <histogram1d path="/" name="Edep" title="Energy deposit">
    <annotation>
      <item key="axis_x.title" value=""/>
    </annotation>
    <axis direction="x" numberOfBins="100" min="0" max="10000"/>
    <statistics entries="5000">
      <statistic direction="x" mean="4991.163651377722089819144" rms="25.00646158764153881293169"/>
    </statistics>
    <data1d>
      <bin1d binNum="44" entries="1" height="1" error="1" weightedMean="4451.520478311058468534611"/>
      <bin1d binNum="45" entries="3" height="3" error="1.732050807568877193176604"
          weightedMean="4576.961659167744073783979" weightedRms="22.32855470030648348256364"/>
      <bin1d binNum="46" entries="2" height="2" error="1.414213562373095145474622"
          weightedMean="4621.246948266485560452566" weightedRms="15.71296641756487666441444"/>
      <bin1d binNum="47" entries="6" height="6" error="2.449489742783177881335632"
          weightedMean="4756.059668558203156862874" weightedRms="25.49848472499792961798448"/>
      <bin1d binNum="48" entries="55" height="55" error="7.416198487095662983392685"
          weightedMean="4866.724766711570737243164" weightedRms="28.30993586766123470965795"/>
      <bin1d binNum="49" entries="4921" height="4921" error="70.14983962918233828531811"
          weightedMean="4993.326701671887349220924" weightedRms="12.66430601818946577452607"/>
      <bin1d binNum="50" entries="12" height="12" error="3.464101615137754386353208"
          weightedMean="5002.203202368423262669239" weightedRms="0.5731653327696435429317035"/>
    </data1d>
</aida>
```

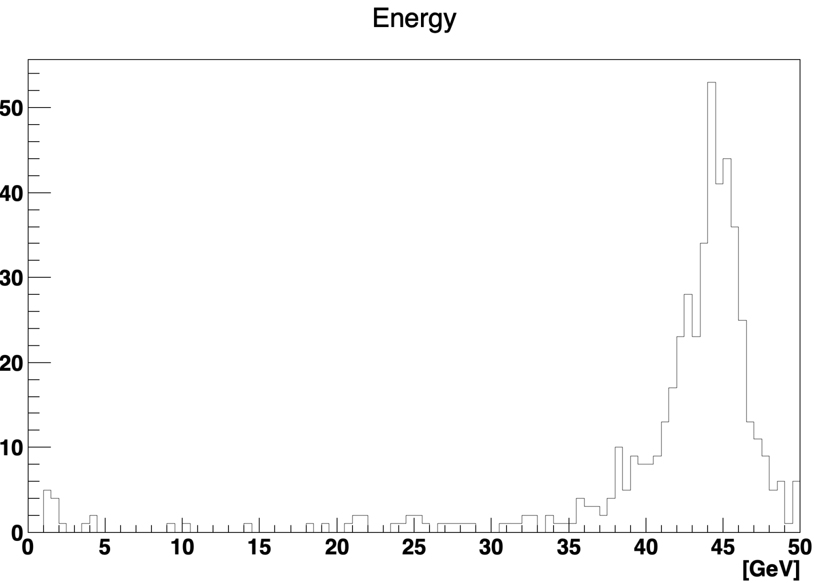


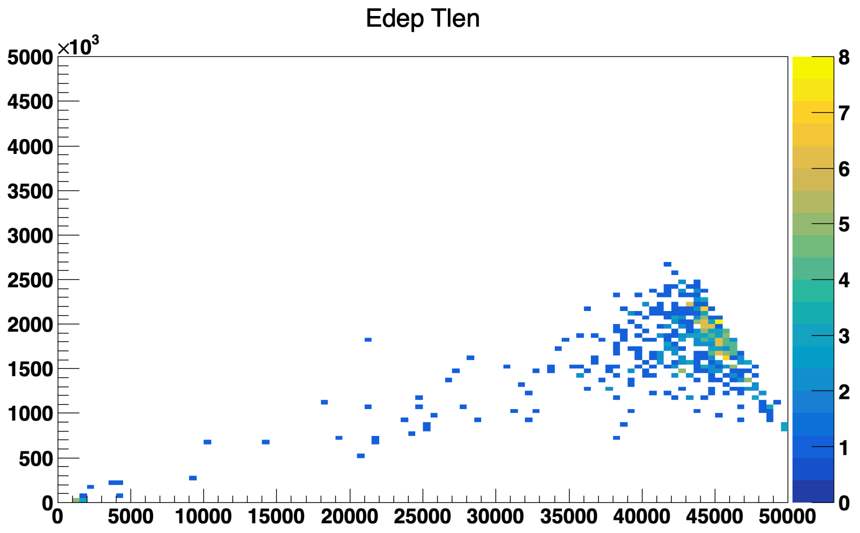
A1: #class tools::wcsv::ntuple

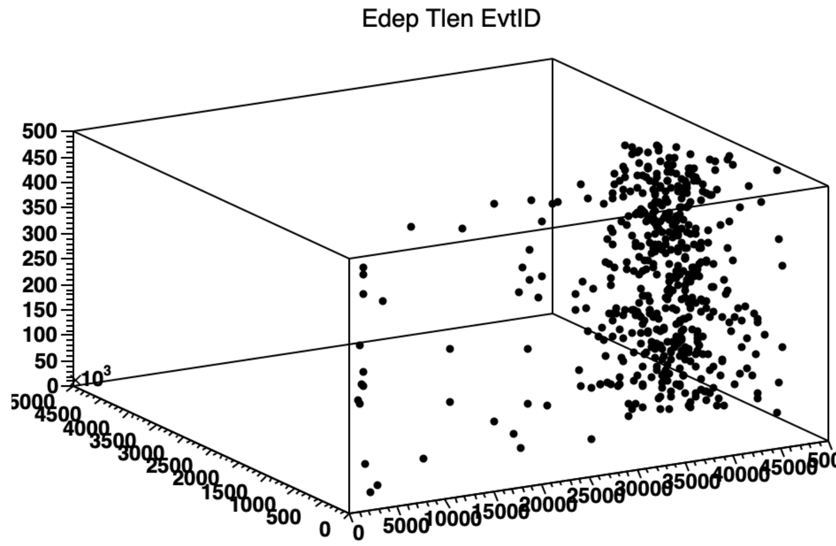
	A
1	#class tools::wcsv::ntuple
2	#title Ntuple
3	#separator 44
4	#vector_separator 59
5	#column double Energy
6	4986.25
7	4988.8
8	5000
9	4996.42
10	4985.11
11	4998.78
12	4988.71
13	5000
14	5000
15	5000
16	5000
17	4979.43
18	4998.82
19	5000
20	4996.43
21	4998.83
22	5000
23	4988.8
24	5000
25	4988.8
26	5000
27	4969.2
28	5000
29	5000
30	5000

More on histograms (1/3)

- It is possible to create 1D (`CreateH1()`), 2D (`CreateH2()`), 3D (`CreateH3()`) histograms as well as 1D (`CreateP1()`) and 2D (`CreateP2()`) *profile* histograms
- **IDENTIFIERS**: each histo ID is automatically generated when the histo is created and its value returned by creating function (histo names are not related to IDs and cannot be used for filling)
  - Default start value is 0, it can be changed with `G4AnalysisManager::SetFirstHistoId(G4int)`
  - IDs for H1, H2, H3, P1 and P2 are defined independently







More on histograms

It is possible to access histogram objects via the `G4AnalysisManager`

RunAction.cc - accessing histogram(s) mean and rms

```
void RunAction::EndOfRunAction(const G4Run* /*run*/) {

    auto analysisManager = G4AnalysisManager::Instance();

   G4cout << G4endl << " ----> print histograms statistic ";
    if(isMaster) {
      G4cout << "for the entire run " << G4endl << G4endl;
    }
    else {
      G4cout << "for the local thread " << G4endl << G4endl;
    }
    G4cout << " Edep : mean = "
       << G4BestUnit(analysisManager->GetH1(0)->mean(),
"Energy")
       << " rms = "
       << G4BestUnit(analysisManager->GetH1(0)->rms(),
"Energy") << G4endl;

}
```

Terminal - Geant4 execution on 2 *threads*

```
*********************************************************
 Geant4 version Name: geant4-11-01 [MT]   (9-December-2022)
  << in Multi-threaded mode >>
     Copyright  : Geant4 Collaboration
     References : NIM A 506 (2003), 250-303
                : IEEE-TNS 53 (2006), 270-278
                : NIM A 835 (2016), 186-225
                  WWW : http://geant4.org/
*********************************************************



// some output



G4WT0 >  ----> print histograms statistic for the local thread
G4WT0 >  Edep : mean = 41.117 MeV rms = 9.2968 MeV
G4WT1 >  ----> print histograms statistic for the local thread
G4WT1 >  Edep : mean = 41.906 MeV rms = 7.36116 MeV
----> print histograms statistic for the entire run
Edep : mean = 41.5036 MeV rms = 8.41347 MeV
```

More on histograms

Additional histogram properties can be defined via the `G4AnalysisManager`

- Unit: if defined, filled values are automatically converted to it
  `analysisManager->CreateH1("Tlen","Tracks length",100,0.*km,5.*km,"km")`
- Function: if defined, the function is automatically executed on the filled value (can be `log`, `log10`, `exp`)
  `analysisManager->CreateH1("Tlen","Tracks length",100,0.*km,5.*km, “km”,”exp”)`
  (if both unit and function are defined, unit is applied first)
- Binning scheme: default is linear, possible to set it to logarithm (`lin`, `log`)
  `analysisManager->CreateH1("Tlen","Tracks length”,100,0.1*km,5.*km,“none”,”none”,”log”)`
- Alternative binning scheme: alternative constructor available to set a non-equidistant binning scheme using a vector of bin edges

```
G4int CreateH1(const G4String& name, const G4String& title,
               const std::vector<G4double>& edges,
               const G4String& unitName = "none",
               const G4String& fcnName = "none");
```

More on ntuples

- **IDENTIFIERS**: ntuple and ntuple column IDs are automatically generated and returned by `G4AnalysisManager->CreateNtuple()` and `G4AnalysisManager->CreateNtupleXColumn()`
- IDs must be used to fill columns if more than one ntuple is created

```
void EventAction::EndOfEventAction(const G4Event* event){

    auto analysisManager = G4AnalysisManager::Instance();

    //Fill ntuple 0 and 1
    //
    analysisManager->FillNtupleDColumn(0, 0, Edep); //ntuple ID, column ID, value
    analysisManager->FillNtupleDColumn(0, 1, TrackL);
    analysisManager->AddNtupleRow(0);

    analysisManager->FillNtupleDColumn(1, 0, Edep/2.);
    analysisManager->FillNtupleDColumn(1, 1, TrackL/2.);
    analysisManager->AddNtupleRow(1);

}
```

- The following types can be saved in columns: `int (I)`, `float (F)`, `double (D)`, `string (S)`, `std::vector` of `int (I)`, `float (F)`, and `double (D)`

g4analysis - UI commands (1/2)

Several UI commands are available to interact with the G4AnalysisManager in your macro card

- General options and handling histos

```
/analysis/verbose 1                              # Set verbose level
/analysis/h1/create Edep Energy 100 0. 100. GeV # create H1, ID=0
/analysis/h1/set 0 100 0. 50. GeV                # set H1 nbins, min, max
```

(similar commands available for H2, H3, P1 and P2)

- Handling output file

```
/analysis/setFileName name                       # set output file name
/analysis/setHistoDirName histos                 # store hist in directory
/analysis/setNtupleDirName ntuples               # store ntuple in directory
```

g4analysis - UI commands (2/2)

Several UI commands are available to interact with the `G4AnalysisManager` in your macro card

- Histograms control (similar commands available for H2, H3, P1 and P2)

```
/analysis/h1/setAscii id true|false              # activate printing h1 on ASCII file
/analysis/h1/setTitle id title                   # Set title for the 1D histogram
/analysis/h1/setXaxis id title                   # Set x-axis, y-axis title for the 1D histogram
/analysis/h1/setYaxis id title
```

- Batch graphics (similar commands available for H2, H3, P1 and P2)

```
/analysis/h1/setPlotting id true|false           # Automatic plotting of h1
/analysis/h1/setPlottingToAll true|false
```

also available as plain `C++` `analysisManager->SetH1Plotting(id, true)`,
the selected objects will be automatically plotted in a single (.ps) file with page size fixed to A4 format

Recap of Analysis

Geant4 provides a *lightweight* analysis tool

- It handles the histograms and ntuples creation and storage in ROOT, (AIDA) XML, CSV and HDF5 format
- It is fully integrated in the Geant4 framework (units, UI commands)
- It is specifically designed to deliver thread-safe solutions in multi-threaded simulations

All basic and extended examples, as well as several advanced examples, use the `G4AnalysisManager`

- Two examples address data persistence in Geant4 (`examples/extended/analysis`)
  - **AnaEx01**, demonstrates how to use `g4analysis` tools for ntuples and histograms creation
  - **AnaEx02**, achieves same results by linking `ROOT` to a `Geant4` application

Much more information in the Geant4 Analysis Documentation