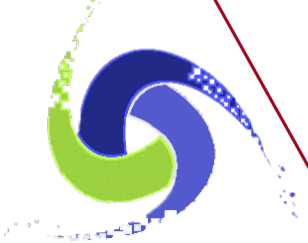


GEANT4
A SIMULATION TOOLKIT
Version 10.5

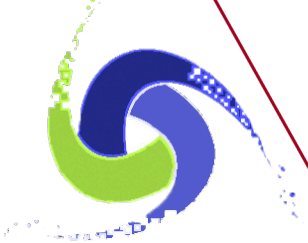
Analysis

Makoto Asai (SLAC)

Geant4 Tutorial Course

Contents

- Introduction on analysis
- Histograms and profile plots
- Ntuples



GEANT4

A SIMULATION TOOLKIT

Version 10.5

Introduction

Introduction

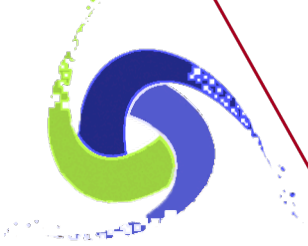
- Geant4 does not provide a complete analysis sub-system
  - Our user community is too heterogeneous
  - Each user group has its own requirements and favorite tools
  - e.g. Python, ROOT in HEP, what is yours ?
- Typical simulation output consists of
  - n-tuple like tables (row: event, column: quantities)
  - Histograms
- Analysis category in Geant4 since December 2011
  - Before it, the analysis code in Geant4 examples used external tools (based on AIDA = Abstract Interfaces for Data Analysis) that had to be linked with the Geant4 application to produce histograms or ntuples
- Area of new developments and improvements: more features are added in each release
  - Example: better MPI (Message Passing Interface) and MT support
- Based on g4tools from inlib/exlib developed by Guy Barrand (LAL, France)
  - http://inexlib.lal.in2p3.fr
  - “Pure header code” - all code is inlined : can be installed on iOS, Android, UNIXes, Windows...

G4Analysis

- A singleton G4AnalysisManager
  - Handles creation of output file(s)
  - Owns and manages histograms and n-tuples
- G4AnalysisManager provides unique interfaces regardless of the choice of output format to write histograms and “flat n-tuples” (i.e. with primitive types).
  - several formats: ROOT, XML AIDA, CSV, HBOOK
- It is integrated in the Geant4 kernel and thread-safe.
  - It provides automatic merging over threads.
- In examples/extended/analysis, 3 examples to demonstrate how to make histograms and ntuples
  - AnaEx01 – use of Geant4 analysis tools
  - AnaEx02 – use of ROOT classes, requires linking with Root libraries
  - AnaEx03 – use of AIDA interface classes, requires linking with an AIDA compliant tool, eg. OpenScientist

Output file

- Depending on selected file format, multiple output files can be produced
  - ROOT
    - All histograms, profiles and n-tuples are written in one file
  - XML (AIDA)
    - The histograms and profiles are written in one file, and each n-tuple is written in a separate file
  - CSV (comma-separated values)
    - Each histogram, profile and n-tuple are written in a separate file
  - File names are generated automatically fileName[_objectName].ext where ext = xml, csv, root



GEANT4

A SIMULATION TOOLKIT

Version 10.5

Histograms and profile plots

Histograms and profile plots

- Selection of output format
  - The generic types are defined in dedicated header files for each output type:
    - g4root.hh, g4csv.hh, g4xml.hh
  - It is recommended to add the selected include in an extra header file MyAnalysis.hh and include this header file in all classes which use g4analysis

MyAnalysis.hh

```
#ifndef MyAnalysis_h
#define MyAnalysis_h 1

#include "g4root.hh"
//#include "g4csv.hh"
//#include "g4xml.hh"
#endif
```

- Three steps
  1. Open a file and define histograms : typically in RunAction::BeginOfRunAction()
  2. Fill values : typically in Run::RecordEvent() or EventAction::EndOfEventAction()
  3. Write and close file : typically in RunAction::EndOfRunAction()

Step 1 : define histograms

```
#include "MyAnalysis.hh"
void MyRunAction::BeginOfRunAction(const G4Run* run)
{
  // Create/get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  analysisManager->SetVerboseLevel(1);

 // Open an output file
  analysisManager->OpenFile("MyApplication");

  // Create histograms
  analysisManager→CreateH1("Edep","Energy deposit", 100, 0., 800*MeV);
  analysisManager→CreateH1("Tlen","Track length", 100, 0., 100*mm);
}
```

Step 2 : fill histograms

```
#include "MyAnalysis.hh"
void MyRun::RecordEvent(const G4Event* anEvent)
{
  auto analysisManager = G4AnalysisManager::Instance();
  auto hce = anEvent->GetHCofThisEvent();
  auto score = (G4THitsMap<G4double>*)(hce->GetHC(collectionID));
  for(auto hItr : *score)
  { analysisManager->FillH1(0, *(hItr.second)); }
}
```

Step 3 : write and close

```
#include "MyAnalysis.hh"
MyRunAction::EndOfRunAction(const G4Run* run)
{
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();

  analysisManager→Write();
  analysisManager→CloseFile();
}
```

Step 3 : Histograms, profile plots

- 1-D, 2-D, 3-D histograms and 1-D, 2-D profile plots are available.
- Histogram identifiers
  - The histogram ID is automatically generated when a histogram is created and it is returned from the “Create” function
  - Note: the histogram names have no relation to the histogram ID which is used at filling
  - The 1D, 2D and 3D histograms IDs are defined independently.
- It is also possible to access directly to a histogram by G4AnalysisManager::GetH1(G4int id).

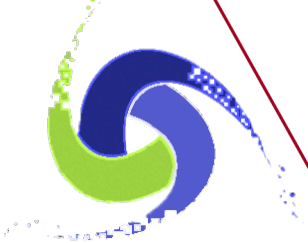
```
G4cout << "Print histograms statistic \n" << G4endl;
G4cout << " EAbs : mean = "
       << analysisManager->GetH1(1)->mean()
       << " rms = "
       << analysisManager->GetH1(1)->rms() << G4endl;
```

Histogram options

- Properties, additional to those defined in g4tools, can be added to histograms via G4AnalysisManager
  - Unit : if defined, all filled values are automatically converted to this defined unit
  - Function : if defined, the function is automatically executed on the filled values (can be log, log10, exp)
    - When a histogram is defined with both unit and function, then the unit is applied first
  - Binning scheme : users can define a non-equidistant binning scheme (passing a vector of bin edges)
- In addition to writing histogram to a file, direct drawing to a Postscript file is available.

```
// Activate plotting of 1D histogram
analysisManager->SetH1Plotting(id, true);
// etc for H2, H3, P1, P2
```

```
/analysis/h1/setPlotting id true|false
/analysis/h1/setPlottingToAll true|false
## etc for h2, h3, p1, p2
```



GEANT4

A SIMULATION TOOLKIT

Version 10.5

Ntuples

Step 1 : define n-tuple

```
#include "MyAnalysis.hh"
void MyRunAction::BeginOfRunAction(const G4Run* run)
{
  // Create/get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  analysisManager->SetVerboseLevel(1);

 // Open an output file
  analysisManager->OpenFile("MyApplication");

  // Creation of ntuple
  analysisManager->CreateNtuple("MyNtuple", "Edep and TrackLength");
  analysisManager->CreateNtupleDColumn("Eabs");
  analysisManager->CreateNtupleDColumn("Labs");
  analysisManager->FinishNtuple();
}
```

Note: available column types:

- integer (I), float (F), double (D), string (S),
- std::vector of integer (I), float (F), double (D)

Step 2 : fill column

```
#include "MyAnalysis.hh"
void MyRun::RecordEvent(const G4Event* anEvent)
{
  auto analysisManager = G4AnalysisManager::Instance();
  auto hce = anEvent->GetHCofThisEvent();
  auto score = (G4THitsMap<G4double>*)(hce->GetHC(collectionID));
  G4double val = 0.;
  for(auto hItr : *score)
  { val += *(hItr.second); }
  analysisManager->FillNtupleDColumn(0, val);
}
```

Step 3 : write and close

```
#include "MyAnalysis.hh"
MyRunAction::EndOfRunAction(const G4Run* run)
{
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();

  analysisManager→Write();
  analysisManager→CloseFile();
}
```