

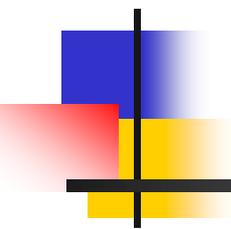


Interaction with the Geant4 kernel – part 2

Luciano Pandola

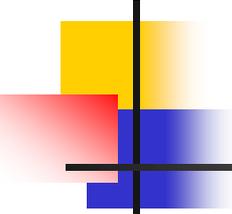
INFN – Laboratori Nazionali del Sud

Partially based on a presentation by G.A.P. Cirrone (INFN-LNS)

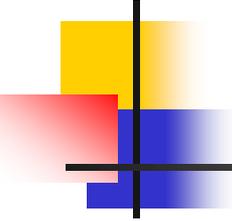


Part V: Write information on output files

Introduction: data analysis with Geant4

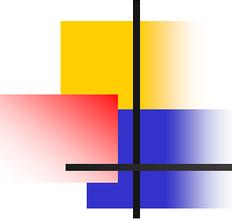


- For a long time, Geant4 did not attempt to provide/support **any data analysis** tools
 - The **focus** was given (and is given) to the **central mission** as a **Monte Carlo simulation** toolkit
 - As a general rule, the **user** is **expected** to provide her/his own **code** to **output results** to an appropriate analysis format
- **Basic classes** for **data analysis** have recently been implemented in Geant4 (g4analysis)
 - Support for **histograms** and **ntuples**
 - Output in **ROOT**, **XML**, **HBOOK** and **CSV** (ASCII)
 - Appropriate only for **easy/quick analysis**: for advanced tasks, the user must write his/her own code and to use an external analysis tool



Introduction: how to write simulation results

- Formatted (= human-readable) **ASCII files**
 - **Simplest** possible approach is **comma-separated values** (.csv) files
 - The resulting files can be opened and analyzed by tools such as: Gnuplot, Excel, OpenOffice, Matlab, Origin, ROOT, PAW, ...
- **Binary files** with complex analysis objects (Ntuples)
 - Allows to **control** what plot you want **with modular choice of conditions** and variables
 - Ex: energy of electrons knowing that (= cuts): (1) position/location, (2) angular window, (3) primary/secondary ...
 - Tools: Root , PAW, AIDA-compliant (PI, JAS3 and OpenScientist)



Output stream (G4cout)

- **G4cout** is a **ostream** object defined by Geant4.
 - The usage of this objects is exactly the **same** as the **ordinary std::cout** except that the output streams will be handled by **G4UImanager**
 - **G4endl** is the equivalent of **std::endl** to end a line
- Output strings may be displayed on another window or stored in a file
- One can also use the file streams (**std::ofstream**) provided by the **C++ libraries**

Output on screen – an example

```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{

    evtNb = eventAction -> Trasporto();

    G4String particleName = aStep -> GetTrack() -> GetDynamicParticle() -> GetDefinition() -> GetParticleName();
    G4String volumeName = aStep -> GetPreStepPoint() -> GetPhysicalVolume() -> GetName();
    G4double particleCharge = aStep -> GetTrack() -> GetDefinition() -> GetAtomicNumber();
    G4double PDG=aStep->GetTrack()->GetDefinition()->GetAtomicMass();

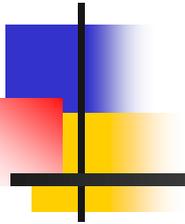
    G4Track* theTrack = aStep->GetTrack();
    G4double kineticEnergy = theTrack -> GetKineticEnergy();
    G4int trackID = aStep -> GetTrack() -> GetTrackID();
    G4double edep = aStep->GetTotalEnergyDeposit();
    G4String materialName = theTrack->GetMaterial()->GetName();

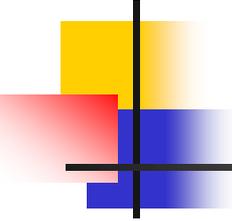
    G4cout      << "Energy deposited--->" << " " << edep << " "
    << "Charge--->" << " " << particleCharge << " "
    << "Kinetic Energy --->" << " " << kineticEnergy << " "
    << G4endl;
```

Output on screen – an example

```
---> Begin of Event: 0
Energia depositata---> 9.85941e-22 Carica---> 6 Energia Cinetica---> 160
Energia depositata---> 8.36876 Carica---> 6 Energia Cinetica---> 151.631
Energia depositata---> 8.63368 Carica---> 6 Energia Cinetica---> 142.998
Energia depositata---> 5.98509 Carica---> 6 Energia Cinetica---> 137.012
Energia depositata---> 4.73055 Carica---> 6 Energia Cinetica---> 132.282
Energia depositata---> 0.0225575 Carica---> 6 Energia Cinetica---> 132.259
Energia depositata---> 1.47468 Carica---> 6 Energia Cinetica---> 130.785
Energia depositata---> 0.0218983 Carica---> 6 Energia Cinetica---> 130.763
Energia depositata---> 5.22223 Carica---> 6 Energia Cinetica---> 125.541
Energia depositata---> 7.10685 Carica---> 6 Energia Cinetica---> 118.434
Energia depositata---> 6.62999 Carica---> 6 Energia Cinetica---> 111.804
Energia depositata---> 6.50997 Carica---> 6 Energia Cinetica---> 105.294
Energia depositata---> 6.28403 Carica---> 6 Energia Cinetica---> 99.0097
Energia depositata---> 5.77231 Carica---> 6 Energia Cinetica---> 93.2374
Energia depositata---> 5.2333 Carica---> 6 Energia Cinetica---> 88.0041
Energia depositata---> 3.9153 Carica---> 6 Energia Cinetica---> 84.0888
Energia depositata---> 14.3767 Carica---> 6 Energia Cinetica---> 69.7121
Energia depositata---> 14.3352 Carica---> 6 Energia Cinetica---> 55.3769
```

Part VI: User-defined sensitive detectors: Hits and Hits Collection

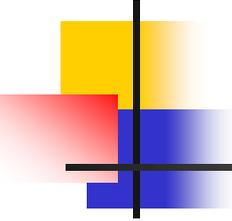




The ingredients of user SD

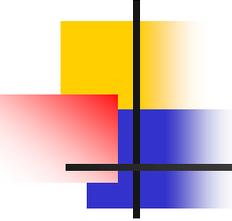
- A **powerful** and **flexible** way of extracting information from the physics simulation is to **define your own SD**
- Derive **your own concrete classes** from the base classes and **customize** them according to your needs

	Concrete class	Base class
Sensitive Detector	MySensitiveDetector	G4VSensitiveDetector
Hit	MyHit	G4VHit
		Template class
Hits collection		G4THitsCollection<MyHit* >



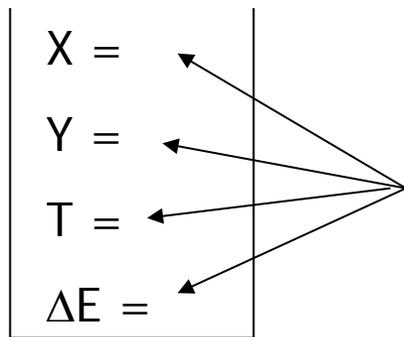
Hit class - 1

- Hit is a **user-defined class** which derives from the base class **G4VHit**. Two **virtual methods**
 - Draw()
 - Print()
- You can **store various types of information** by implementing your own concrete Hit class
- Typically, one may want to record information like
 - Position, time and ΔE of a step
 - Momentum, energy, position, volume, particle type of a given track
 - Etc.



Hit class - 2

A “Hit” is like a “**container**”, a **empty box** which will store the **information** retrieved step by step



The Hit **concrete class** (derived by **G4VHit**) **must** be **written by the user**: the user must decide **which variables and/or information** the hit should **store** and **when** store them

The Hit objects are **created** and **filled** by the **SensitiveDetector** class (invoked at each step in **detectors defined as sensitive**). **Stored** in the “**HitCollection**”, attached to the **G4Event**: can be **retrieved** at the end of the event

Hit class - 3

Example

```
// header file: MyHit.hh
```

```
#include "G4VHit.hh"
```

```
class MyHit : public G4VHit {
```

```
public:
```

```
    MyHit();
```

```
    virtual ~MyHit();
```

```
    ...
```

public methods to
handle data member

```
    inline void SetEnergyDeposit(G4double energy) { energyDeposit = energy; }
```

```
    inline G4double GetEnergyDeposit() { return energyDeposit; }
```

```
    ... // more get and set methods
```

```
private:
```

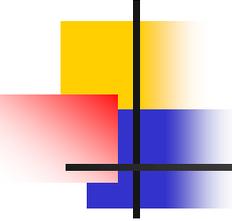
```
    G4double energyDeposit;
```

```
    ... // more data members
```

```
};
```



data member (private)



Geant4 Hits

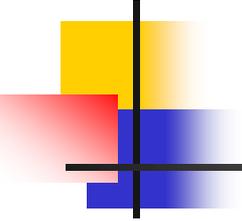
Since in the simulation one may have **different sensitive detectors** in the same setup (e.g. a calorimeter and a Si detector), it is possible to define **many Hit classes** (all derived by **G4VHit**) storing **different information**

X =
Y =
T =
$\Delta E =$

```
Class Hit1 :  
public G4VHit
```

Z =
Pos =
Dir =

```
Class Hit2 :  
public G4VHit
```



Hits Collection - 1

At each step in a **detector** defined as **sensitive**, the method **ProcessHit()** of the user **SensitiveDetector** class is invoked: it must **create**, **fill** and **store** the Hit objects

X = 1
Y = 2
T = 3
$\Delta E = 1$

Step 1

X = 2
Y = 0
T = 3.1
$\Delta E = 2$

Step 2

X = 3
Y = 2
T = 4
$\Delta E = 3$

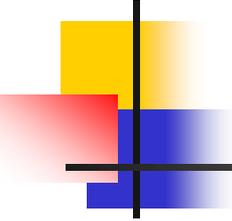
Step 3

.....

X = 3
Y = 2
T = 6
$\Delta E = 1$

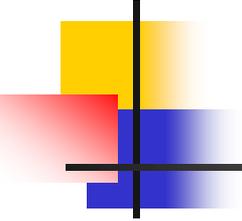
Step N

Hits collection (= vector<Hit>)



Hits Collection - 2

- Once created in the sensitive detectors, objects of the concrete hit class **must be stored** in a **dedicated collection**
 - **Template class** `G4THitsCollection<MyHit>`, which is actually an **array of `MyHit*`**
- The hits collections can be accessed in **different phases** of tracking
 - At the **end of each event**, through the `G4Event` (*a-posteriori event analysis*)
 - During **event processing**, through the Sensitive Detector Manager `G4SDManager` (*event filtering*)



The HCofThisEvent

Remember that you may have **many kinds of Hits**
(and Hits Collections)

X = 1
Y = 2
T = 3
 $\Delta E = 1$

X = 2
Y = 0
T = 3.1
 $\Delta E = 2$

X = 3
Y = 2
T = 4
 $\Delta E = 3$

.....

X = 3
Y = 2
T = 6
 $\Delta E = 1$

Z = 5
Pos =
(0,1,1)
Dir
=(0,1,0)

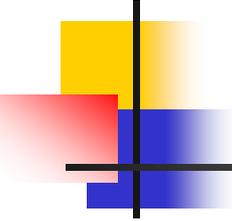
Z = 5.2
Pos =
(0,0,1)
Dir
=(1,1,0)

.....

Z = 5.4
Pos =
(0,1,2)
Dir
=(0,1,1)

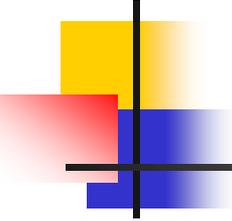
HCofThisEvent

Attached to
G4Event*



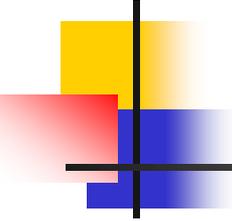
Hits Collections of an event

- A **G4Event** object has a **G4HCofThisEvent** object at the end of the event processing (if it was successful)
 - The **pointer** to the **G4HCofThisEvent** object can be retrieved using the **G4Event::GetHCofThisEvent()** method
- The **G4HCofThisEvent** stores all **hits collections** created within the event
 - Hits collections are **accessible** and can be **processed** e.g. in the **EndOfEventAction()** method of the User **Event Action** class



SD and Hits

- Using **information** from **particle steps**, a sensitive detector either
 - **constructs**, **fills** and **stores** one (or more) **hit object**
 - **accumulates** values to existing hits
- Hits objects can be **filled** with information in the **ProcessHits()** method of the SD concrete user class → **next slides**
 - This method has **pointers** to the current **G4Step** and to the **G4TouchableHistory** of the ReadOut geometry (if defined)



Sensitive Detector (SD)

- A specific feature to Geant4 is that a user can provide his/her **own implementation** of the **detector** and **its response** → **customized**
- To create a **sensitive detector**, **derive** your own **concrete class** from the **G4VSensitiveDetector** abstract base class
 - The principal purpose of the sensitive detector is to **create hit objects**
 - Overload the **following methods** (see also next slide):
 - **Initialize()**
 - **ProcessHits()** (Invoked for **each step** if step starts in logical volume having the SD attached)
 - **EndOfEvent()**

Sensitive Detector

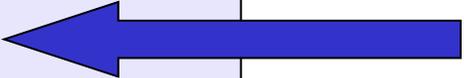
```
class G4VSensitiveDetector {  
public:                                     abstract base class  
    ...  
    virtual void Initialize(G4HCofThisEvent*);  
    virtual void EndOfEvent(G4HCofThisEvent*);  
protected:  
    virtual G4bool ProcessHits(G4Step* ,  
                                G4TouchableHistory*) = 0;  
    ...  
};
```

pure virtual method →

```
// header file: MySensitiveDetector.hh  
#include "G4VSensitiveDetector.hh"
```

```
...  
class MySensitiveDetector : public G4VSensitiveDetector {  
public:  
    MySensitiveDetector(G4String name);  
    virtual ~MySensitiveDetector();  
  
    virtual void Initialize(G4HCofThisEvent*HCE);  
    virtual G4bool ProcessHits(G4Step* step,  
                                G4TouchableHistory* ROhist);  
    virtual void EndOfEvent(G4HCofThisEvent*HCE);  
private:  
    MyHitsCollection * hitsCollection;  
    G4int collectionID;  
};
```

User
concrete
SD class



SD implementation: constructor

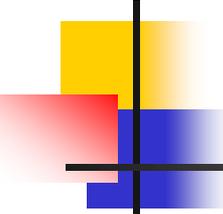
- Specify a **hits collection** (by its unique name) for each type of hits considered in the sensitive detector:
 - Insert the **name(s)** in the **collectionName vector**

```
MySensitiveDetector::MySensitiveDetector(G4String detectorUniqueName)
    : G4VSensitiveDetector(detectorUniqueName),
      collectionID(-1) {
    collectionName.insert("collection_name");
}
```

Base class



```
class G4VSensitiveDetector {
    ...
protected:
    G4CollectionNameVector collectionName;
    // This protected name vector must be filled in
    // the constructor of the concrete class for
    // registering names of hits collections
    ...
};
```



SD implementation: Initialize()

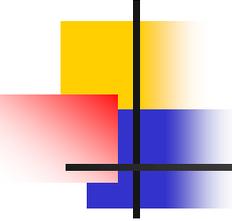
- The `Initialize()` method is invoked at the **beginning** of each event
- Construct all hits collections and insert them in the `G4HCofThisEvent` object, which is passed as argument to `Initialize()`
 - The `AddHitsCollection()` method of `G4HCofThisEvent` requires the **collection ID**
- The unique collection ID can be obtained with `GetCollectionID()`:
 - `GetCollectionID()` cannot be invoked in the constructor of this SD class (It is required that the SD is instantiated and registered to the SD manager first).
 - Hence, we defined a private data member (`collectionID`), which is set at the first call of the `Initialize()` function

```
void MySensitiveDetector::Initialize(G4HCofThisEvent*HCE) {  
    if(collectionID < 0)  
        collectionID = GetCollectionID(0); // Argument : order of collect.  
                                           // as stored in the collectionName  
    hitsCollection = new MyHitsCollection  
        (SensitiveDetectorName, collectionName[0]);  
    HCE -> AddHitsCollection(collectionID, hitsCollection);  
}
```

SD implementation: ProcessHits()

- This **ProcessHits()** method is invoked for **every step** in the volume(s) which **hold a pointer to this SD** (= each volume defined as "**sensitive**")
- The **main mandate** of this method is to **generate hit(s)** or to accumulate data to existing hit objects, by **using information** from the current step
 - Note: Geometry information must be derived from the "**PreStepPoint**"

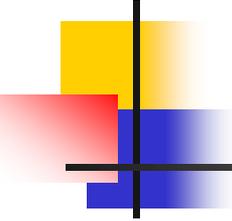
```
G4bool MySensitiveDetector::ProcessHits(G4Step* step,
                                        G4TouchableHistory*ROhist) {
    MyHit* hit = new MyHit(); // 1) create hit
    ...
    // some set methods, e.g. for a tracking detector:
    G4double energyDeposit = step -> GetTotalEnergyDeposit(); // 2) fill hit
    hit -> SetEnergyDeposit(energyDeposit); // See implement. of our Hit class
    ...
    hitsCollection -> insert(aHit); // 3) insert in the collection
    return true;
}
```



SD implementation: EndOfEvent()

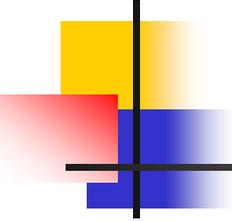
- This `EndOfEvent()` method is invoked at the end of each event.
 - Note is invoked **before** the `EndOfEvent` function of the **`G4UserEventAction`** class

```
void MySensitiveDetector::EndOfEvent(G4HCofThisEvent* HCE) {  
}
```



Processing hit information - 1

- Retrieve the pointer of a hits collection with the `GetHC()` method of `G4HCofThisEvent` collection using the **collection index** (a `G4int` number)
- **Index** numbers of a hit collection are **unique** and don't change for a run. The number **can be obtained** by `G4SDManager::GetCollectionID("name");`
- Notes:
 - if the collection(s) are **not created**, the pointers of the collection(s) are NULL: **check** before trying to access it
 - Need an **explicit cast** from `G4VHitsCollection` (see code)



Processing hit information - 2

- **Loop** through the entries of a hits collection to **access individual hits**
 - Since the HitsCollection is a vector, you can use the **[] operator** to get the hit object corresponding to a **given index**
- **Retrieve** the information **contained in this hit** (e.g. using the **Get/Set methods** of the concrete user Hit class) and **process it**
- **Store** the output in analysis objects

Process hit: example

```
void MyEventAction::EndOfEventAction(const G4Event* event) {  
    // index is a data member, representing the hits collection index of the  
    // considered collection. It was initialized to -1 in the class constructor  
    if(index < 0) index =  
        G4SDManager::GetSDMpointer() -> GetCollectionID("myDet/myColl");  
    G4HCofThisEvent* HCE = event-> GetHCofThisEvent();  
    MyHitsCollection* hitsColl = 0;  
    if(HCE) hitsColl = (MyHitsCollection*)(HCE->GetHC(index));  
    if(hitsColl) {  
        int numberHits = hitsColl->entries();  
        for(int i1= 0; i1 < numberHits ; i1++) {  
            MyHit* hit = (*hitsColl)[i1];  
            // Retrieve information from hit object, e.g.  
            G4double energy = hit -> GetEnergyDeposit;  
            ... // Further process and store information  
        }  
    }  
}
```

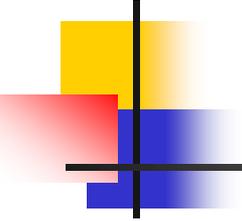
retrieve index

retrieve all hits collections

retrieve hits collection by index

cast

loop over individual hits, retrieve the data



The HCofThisEvent

Remember that you may have **many kinds of Hits**
(and Hits Collections)

X = 1
Y = 2
T = 3
 $\Delta E = 1$

X = 2
Y = 0
T = 3.1
 $\Delta E = 2$

X = 3
Y = 2
T = 4
 $\Delta E = 3$

.....

X = 3
Y = 2
T = 6
 $\Delta E = 1$

Z = 5
Pos =
(0,1,1)
Dir
=(0,1,0)

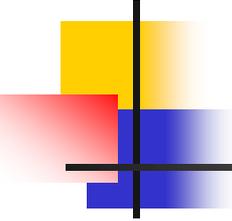
Z = 5.2
Pos =
(0,0,1)
Dir
=(1,1,0)

.....

Z = 5.4
Pos =
(0,1,2)
Dir
=(0,1,1)

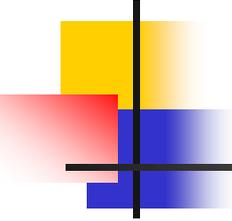
HCofThisEvent

Attached to
G4Event*



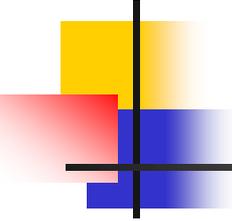
Recipe and strategy - 1

- Create your **detector geometry**
 - Solids, logical volumes, physical volumes
- Implement a **sensitive detector** and assign an instance of it to the ***logical volume*** of your geometry set-up
 - Then this volume becomes “**sensitive**”
 - Sensitive detectors are **active for each particle steps**, if the step starts in this volume



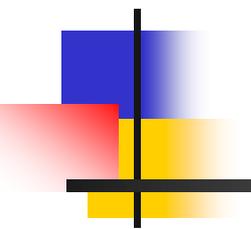
Recipe and strategy - 2

- Create **hits objects** in your sensitive detector using information from the particle step
 - You need to **create the hit class(es)** according to **your requirements**
- **Store** hits in hits collections (automatically associated to the **G4Event** object)
- Finally, **process the information** contained in the hit in user action classes (e.g. **G4UserEventAction**) to obtain **results** to be stored in the analysis object



Hands-on session

- Task5
 - Task5d (optional): User custom hit class
- Task6
- <http://geant4.lngs.infn.it/GSSI2016/task5>
- <http://geant4.lngs.infn.it/GSSI2016/task6>



Backup

To write a new ASCII file: a recipe - 1

- Add to the include list of your class the `<fstream>` header file
 - This will allow to use the C++ libraries for stream on file
- Put into the **class declaration** (file .hh) an ofstream (=output file stream) object (or pointer):
`std::ofstream myFile;`
 - In this way, the file object will be **visible in all methods** of the class
- **Open the file**, in the class constructor, or into a specific method:
`myFile.open("filename.out",
std::ios::trunc);`
 - To **append** data to an existing file, you must specify `std::ios::app`

To write a new ASCII file: a recipe - 2

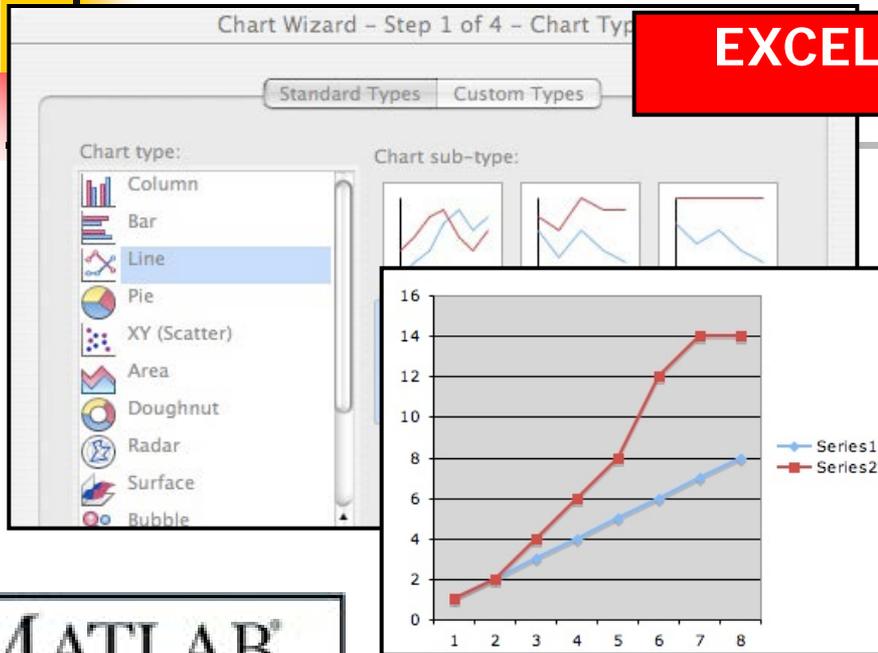
- Inside a regularly called method (e.g. inside a virtual method of an User Class), where appropriate, **write your data** (i.e. **G4double**, **G4int**, **G4String**,...) to the file, in the same fashion of **G4cout**:

```
if (myFile.is_open()) // Check that file is opened
{
    myFile << kineticEnergy/MeV << " " << dose << G4endl;
    ...
}
```

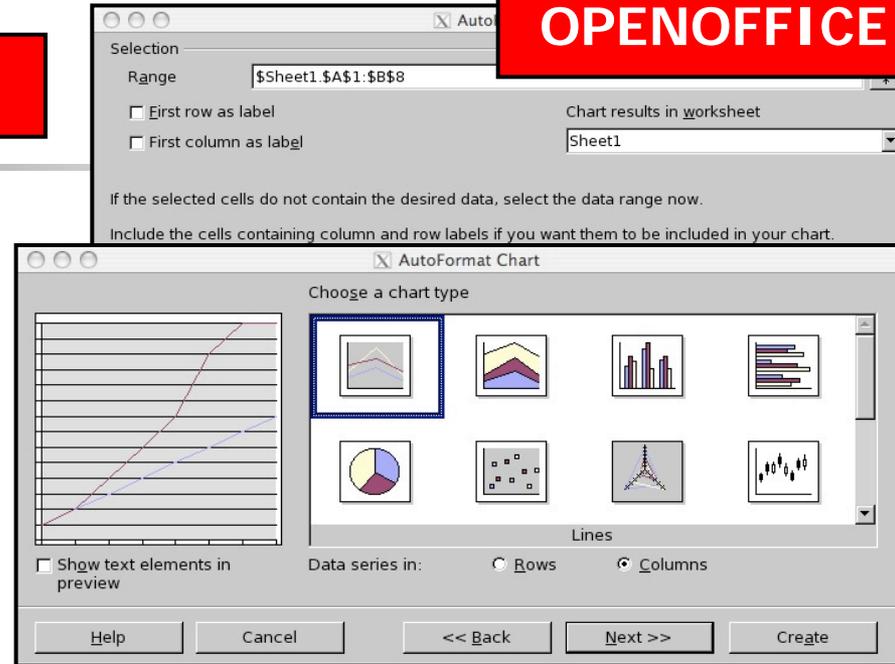
- This could be for instance the **EndOfEventAction()** of the **G4UserEventAction** user class
- Finally **close the file**, in the class destructor, or into a specific method: **myFile.close();**

Plotting with tools

EXCEL

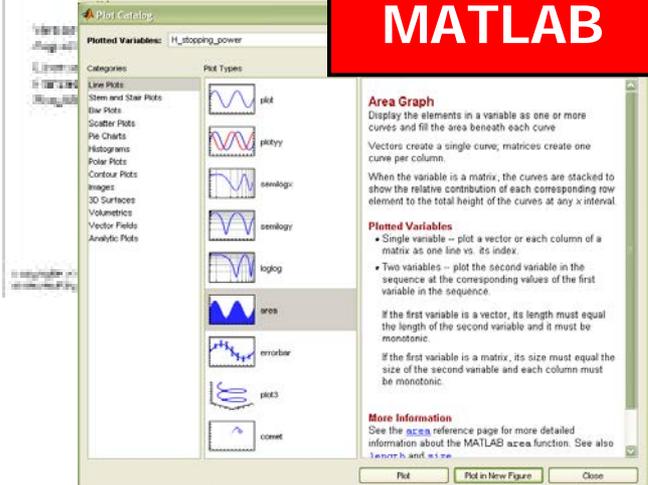


OPENOFFICE



MATLAB

MATLAB



GNU PLOT

