提　纲

- 一、物理过程
- 二、用户定义类
- 三、SteppingAction Class
- 四、EventAction Class
- 五、RunAction Class
- 六、几类UserAction调用关系

一、物理过程

Geant4以轨迹点的方式统一对待所有的物理过程。每一物理过程由两类重要的方法：GetPhysicalInteractionLength(GPIL)方法和DoIt方法。GPIL方法根据计算以截面信息为基础的相互作用几率给出当前时空点和下一时空点之间的步长。之后DoIt方法处理相互作用中的具体详细信息，如改变粒子能量、动量、动量方向、位置坐标以及是否需要产生次级粒子。

物理过程描述粒子与物质的相互作用，Geant4提供了7大类物理过程：

- 电磁相互作用
- 强相互作用
- 输运过程
- 衰变过程
- 光学过程
- 光轻子—强子相互过程
- 参数作用

- Multiple scattering
- Bremsstrahlung
- Ionisation
- Annihilation
- Photoelectric effect
- Compton scattering
- Rayleigh effect
- γ conversion
- e^+e^- pair production
- Synchrotron radiation
- Transition radiation
- Cherenkov
- Refraction
- Reflection
- Absorption
- Scintillation
- Fluorescence
- Auger

- *G4VuserPhysicsList*是Geant4提供的强制基类之一，用户必须在该类中指定模拟所参与的粒子种类和这些粒子参与的物理过程（包括次级粒子的）。用户使用时必须从该基类中派生用户自己所需要的类，并在用户的派生类中执行虚拟方法ConstructProcess().

```
#ifndef ExN01PhysicsList_h
#define ExN01PhysicsList_h 1

#include "G4VUserPhysicsList.hh"
#include "globals.hh"

class ExN01PhysicsList: public G4VUserPhysicsList
{
  public:
    ExN01PhysicsList();
    ~ExN01PhysicsList();
  protected:
    // Construct particle and physics process
    void ConstructParticle();
    void ConstructProcess();
    void SetCuts();
          };
#endif
```

基类 公有继承

在源文件中查找物理过程：

Linux命令find：

find /path/../ －name thefilename

帮助命令：man find

查找到对应的文件夹 Geant4定义的所有过程

按分类放在对应的文件夹下

二、用户可选类

Geant4为用户提供5类虚函数类，用户可以通过重载在模拟的各个阶段对程序控制。每一执行类的方法都提供有默认的实例化，用户可以继承并执行所需的方法。这5种可选类分别为：

Action classes

Invoked during the execution loop

- G4VUserPrimaryGeneratorAction
- G4UserRunAction
- G4UserEventAction
- G4UserStackingAction
- G4UserTrackingAction
- G4UserSteppingAction

三、SteppingAction Class

G4UserSteppingAction的执行发生在粒子输运的每一步，通过其成员函数

Virtual void UserSteppingAction（const G4Step*）{}

可以获得粒子输运过程中的每一步的信息

同样，用户必须从基类G4UserSteppingAction中派生出用户自己的类

```
#ifndef ExN03SteppingAction_h
#define ExN03SteppingAction_h 1

#include "G4UserSteppingAction.hh"

class ExN03DetectorConstruction;
class ExN03EventAction;

class ExN03SteppingAction : public G4UserSteppingAction
{
  public:
    ExN03SteppingAction(ExN03DetectorConstruction*,
ExN03EventAction*);
   ~ExN03SteppingAction();

    void UserSteppingAction(const G4Step*);

  private:
    ExN03DetectorConstruction* detector;
    ExN03EventAction*          eventaction;
};

#endif
```

基类 公有继承

在源文件中查找G4Step.hh 查找其公有成员函数

Linux命令find：

find /path/../ －name G4Step.hh

G4Step.hh中的共有成员函数可以在UserSteppingAction方法中调用，如：

```
G4double edep = aStep->GetTotalEnergyDeposit();
G4double  stepl = aStep->GetStepLength();
```

G4Step类

```
class G4Step
{
 public:
   G4Step();
   ~G4Step();

public:
// Get/Set functions
// currnet track
   G4Track* GetTrack() const;
   void SetTrack(G4Track* value);
// step points
   G4StepPoint* GetPreStepPoint() const;
```

G4Step类

- void SetPreStepPoint(G4StepPoint* value);
- G4StepPoint* GetPostStepPoint() const;
- void SetPostStepPoint(G4StepPoint* value);
- // step length
- G4double GetStepLength() const;
- void SetStepLength(G4double value);
- // total energy deposit
- G4double GetTotalEnergyDeposit() const;
- void SetTotalEnergyDeposit(G4double value);
- // cotrole flag for stepping
- G4SteppingControl GetControlFlag() const;
- void SetControlFlag(G4SteppingControl StepControlFlag);

G4Step类

- // difference of position, time, momentum and energy
- G4ThreeVector GetDeltaPosition() const;
- G4double GetDeltaTime() const;
- G4ThreeVector GetDeltaMomentum() const;
- G4double GetDeltaEnergy() const;
- // manipulation of total energy deposit
- void AddTotalEnergyDeposit(G4double value);
- void ResetTotalEnergyDeposit();
- // manipulation of non-ionizng energy deposit
- void AddNonIonizingEnergyDeposit(G4double value);
- void ResetNonIonizingEnergyDeposit();
- // Get/Set/Clear flag for initial/last step

G4Step类

```
    G4bool IsFirstStepInVolume() const;
    G4bool IsLastStepInVolume() const;
    void SetFirstStepFlag();
    void ClearFirstStepFlag();
    void SetLastStepFlag();
    void ClearLastStepFlag();
  // Other member functions
    void InitializeStep( G4Track* aValue );
  // initiaize contents of G4Step
    void UpdateTrack( );
  // update track by using G4Step information
    void CopyPostToPreStepPoint( );
  // copy PostStepPoint to PreStepPoint
```

G4Step类

```
    G4Polyline* CreatePolyline () const;
   // for visualization
protected:
// Member data
   G4double fTotalEnergyDeposit;
// Accummulated total energy desposit in the current Step
  G4double fNonIonizingEnergyDeposit;
 // Accummulated non-ionizing energy desposit in the current Step
private:
// Member data
   G4StepPoint* fpPreStepPoint;
   G4StepPoint* fpPostStepPoint;
   G4double fStepLength;
     // Step length which may be updated at each invocation of
     // AlongStepDoIt and PostStepDoIt
   G4Track* fpTrack;
G4SteppingControl fpSteppingControlFlag;
    // A flag to control SteppingManager behavier from process
  // flag for initial/last step
   G4bool fFirstStepInVolume;
   G4bool fLastStepInVolume;
// Secondary buckets
public:
   G4TrackVector* GetSecondary() const;
   G4TrackVector* GetfSecondary();
   G4TrackVector* NewSecondaryVector();
   void DeleteSecondaryVector();
```


```
    void SetSecondary( G4TrackVector* value);
  private:
    G4TrackVector* fSecondary;

    // Prototyping implementation of smooth representation of curved
    // trajectories. (jacek 30/10/2002)
  public:
    // Auxiliary points are ThreeVectors for now; change to
    // G4VAuxiliaryPoints or some such (jacek 30/10/2002)
    void
  SetPointerToVectorOfAuxiliaryPoints( std::vector<G4ThreeVector>*
  theNewVectorPointer ) {
      fpVectorOfAuxiliaryPointsPointer = theNewVectorPointer;
    }
    std::vector<G4ThreeVector>*
  GetPointerToVectorOfAuxiliaryPoints() const {
      return fpVectorOfAuxiliaryPointsPointer;
    }
  private:
    // Explicity including the word "Pointer" in the name as I keep
    // forgetting the * (jacek 30/10/2002)
    std::vector<G4ThreeVector>* fpVectorOfAuxiliaryPointsPointer;

  };
  #include "G4Step.icc"
  #endif
```

四、EventAction Class

G4UserEventAction有两个虚拟方法：

- BeginOfEventAction（）
- EndOfEventAction（）

顾名思义，它们的调用发生在每一Event的开始和结束。通过该类可以对Event做简单的分析。如果需要，用户甚至可以在每一Run中挑出并保存所需的Event。（数据量过大的话可以考虑文本输出保存）

基类 公有继承

```
class ExN03EventAction : public G4UserEventAction
{
 public:
   ExN03EventAction(ExN03RunAction*);
  ~ExN03EventAction();
 public:
   void  BeginOfEventAction(const G4Event*);
   void    EndOfEventAction(const G4Event*);

private:
   ExN03RunAction*  runAct;
    G4double  EnergyAbs, EnergyGap;
   G4double  TrackLAbs, TrackLGap;
   G4int     printModulo;
    ExN03EventActionMessenger*  eventMessenger;
};
```

在源文件中查找G4Event.hh 查找其公有成员函数

Linux命令find：

find /path/../ －name G4Event.hh

G4Event.hh中的共有成员函数可以在

- Void BeginOfEventAction(const G4Event*)
- void EndOfEventAction(const G4Event*)

方法中调用。

五、RunAction Class

- G4UserRunAction有三个虚拟方法：
  - GenerateRun()
  - BeginOfRunAction()
  - EndOfRunAction()

其中GenerateRun()是在BeamOn()开始时候调用，而BeginOfRunAciont()是在每一Run开始时候调用，通常可以设置Run的ID号(做输出)、初始化histograms、Sensitive detector的条件限制设置。通常在EndOfRunAction方法中可以对输出结果进行存储、打印输出等操作。

```
#ifndef ExN03RunAction_h
#define ExN03RunAction_h 1

#include "G4UserRunAction.hh"
#include "globals.hh"
class G4Run;                                    基类  公有继承

class ExN03RunAction : public G4UserRunAction
{
  public:
    ExN03RunAction();
   ~ExN03RunAction();
public:
    void BeginOfRunAction(const G4Run*);
    void   EndOfRunAction(const G4Run*);
};
#endif
```

在源文件中查找G4Run.hh 查找其公有成员函数

Linux命令find：

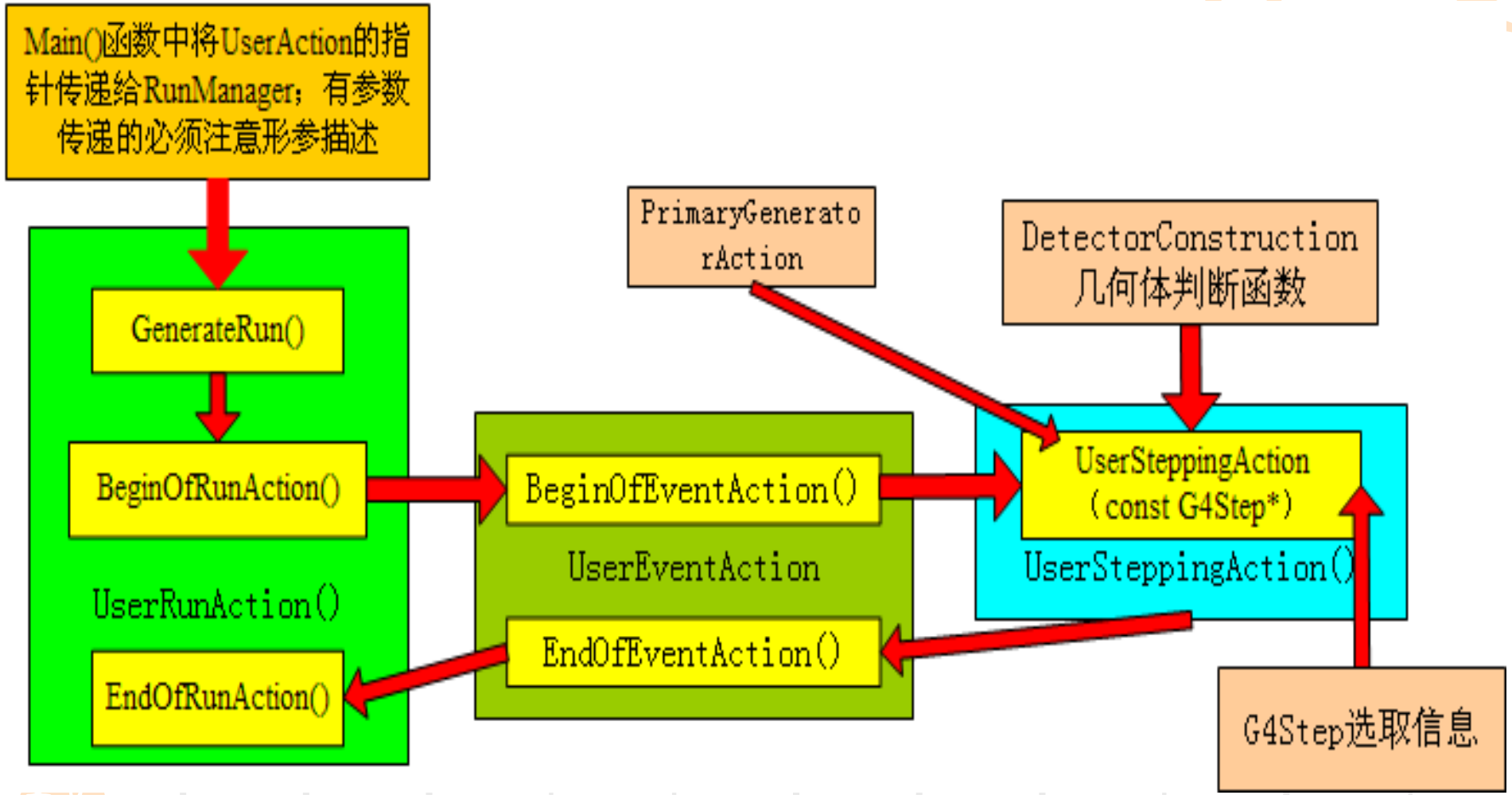
find /path/../ －name G4Run.hh

G4Event.hh中的共有成员函数可以在

- Void BeginOfRunAction(const G4Run*)
- void EndOfRunAction(const G4Run*)

方法中调用。

六、几类UserAction调用关系



各个类之间通过公有函数进行相互调用

例如问题：想要收集粒子在某个几何体A中的**沉积能量**；通过UserAction方法实现！

实现方案分析：

考虑模拟过程：每次Run事件中有多个Event事件，每个Event事件中又包括多个Step；沉积能量为粒子在A中沉积所有能量之和。

在粒子的每次Step模拟中，程序都会调用SteppingAction类中的

`void UserSteppingAction(const G4Step*);`方法

在SteppingAction中通过G4Step判断该Step是否在几何体A中发生，若是，则从G4Step提取所需的信息如沉积能量、粒子数等。

对于某单个粒子，它可能在A中沉积一次能量，也可能沉积多次能量，也可能不沉积能量仅仅穿过A，也可能不穿过A；因此，以Event分类沉积能量较为方便：先得到每个Event沉积能量，然后相加就得到总沉积能量。

每一次Event中，可能会出现很多个Step，在每次Event结束时候，通过EventAction的

void　EndOfEventAction(const G4Event*);方法

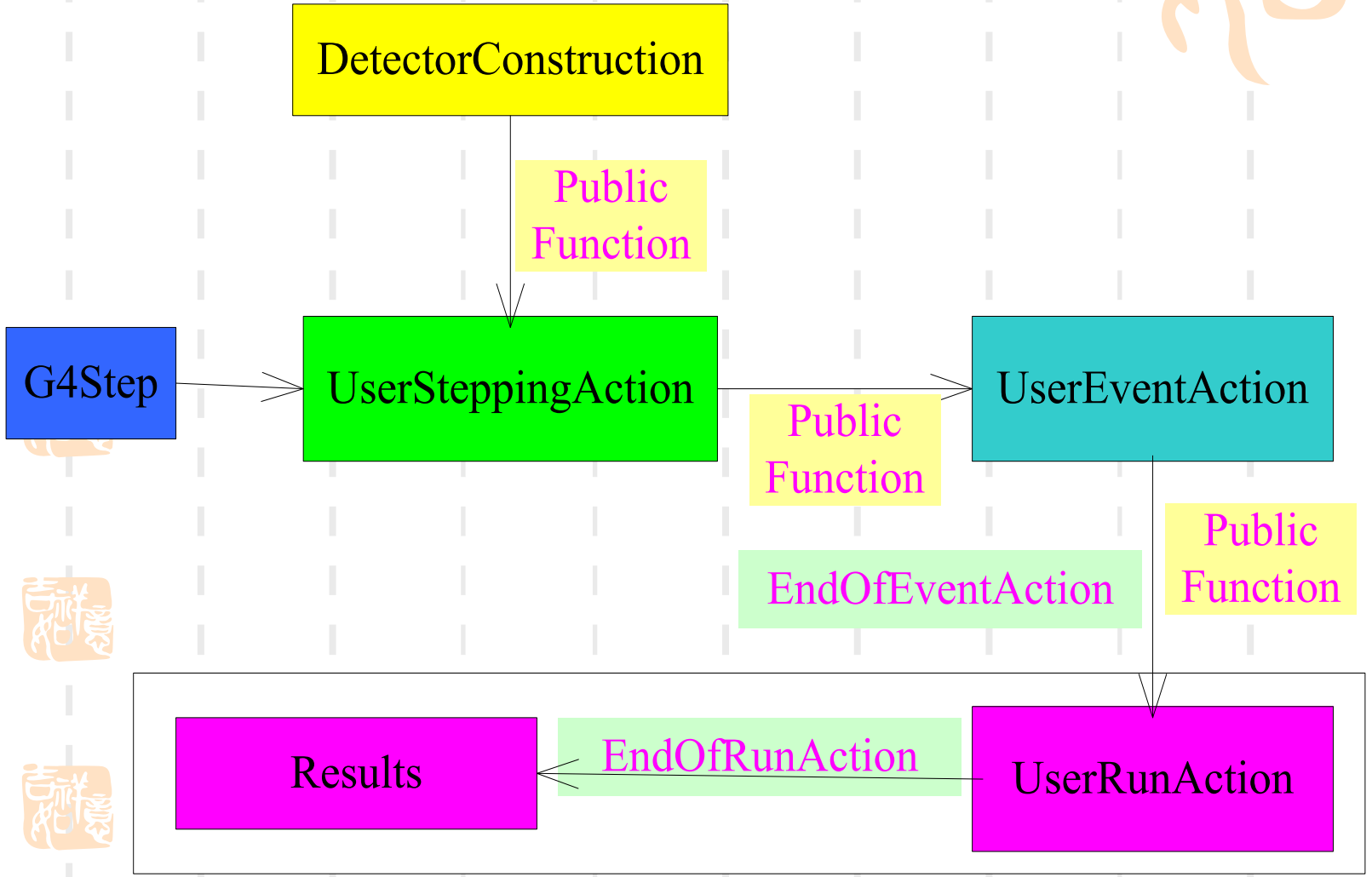
收集每次Event中的粒子沉积能量

每次Run中，可能会有（大多数都会有吧，模拟1个粒子似乎没啥意义！）很多个Event，通过RunAction的

Void EndOfRunAction(const G4Run*)；方法

收集所有Event的信息

类引用关系



在DetectorConstruction.hh中加入如下函数，

```
Public：
const G4VPhysicalVolume* GetA() {return physA;};
```

其中，physA为几何体A的PhysicalVolume指针；

在EventAction.cc的BeginOfEventAction()方法中可以实现变量的初始化；
同样，在RunAction的BeginOfEventAction()方法中实现变量初始化；

在EventAction.hh中加入如下函数，

```
Public:
void AddAbs(G4double de) {EnergyAbs += de;};
```

其中，EnergyAbs为private成员变量；

在RunAction.hh中加入如下函数，

```
Public:
void AddE (G4double de) {AEnergyAbs += de;};
```

其中，EnergyAbs为private成员变量；

UserSteppingAction.hh

```
#include "G4UserSteppingAction.hh"
class ExN03DetectorConstruction;
class ExN03EventAction;
class ExN03SteppingAction : public G4UserSteppingAction
{
  public:
ExN03SteppingAction(ExN03DetectorConstruction*, ExN03EventAction*);
   ~ExN03SteppingAction();
 public：
   void UserSteppingAction(const G4Step*);

  private:
   ExN03DetectorConstruction* detector;
   ExN03EventAction*          eventaction;
};
```

UserSteppingAction.cc

```
#include "ExN03SteppingAction.hh"

#include "ExN03DetectorConstruction.hh"
#include "ExN03EventAction.hh"

#include "G4Step.hh"

ExN03SteppingAction::ExN03SteppingAction(ExN03DetectorConstruction
* det, ExN03EventAction* evt):detector(det), eventaction(evt)
{ }

ExN03SteppingAction::~ExN03SteppingAction()
{ }

void ExN03SteppingAction::UserSteppingAction(const G4Step* aStep)
{
 G4VPhysicalVolume* volume
  = aStep->GetPreStepPoint()->GetTouchableHandle()->GetVolume();

 G4double edep = aStep->GetTotalEnergyDeposit();

  if (volume == detector->GetA ()) eventaction->AddAbs(edep);
}
```

UserEventAction.hh

```
class ExN03EventAction : public G4UserEventAction
{
 public:
   ExN03EventAction(ExN03RunAction*);
  ~ExN03EventAction();

 public:
   void  BeginOfEventAction(const G4Event*);
   void    EndOfEventAction(const G4Event*);

 void AddAbs(G4double de) {EnergyAbs += de;};

 private:
   ExN03RunAction*  runAct;
   G4double  EnergyAbs;
};
```

UserEventAction.cc

```
ExN03EventAction::ExN03EventAction(ExN03RunAction* run)
:runAct(run)
{}
ExN03EventAction::~ExN03EventAction()
{}

void ExN03EventAction::BeginOfEventAction(const G4Event* evt)
{
 G4int evtNb = evt->GetEventID();
   G4cout << "\n---> Begin of event: " << evtNb << G4endl;

// initialisation per event
 EnergyAbs = 0.;
}

void ExN03EventAction::EndOfEventAction(const G4Event* evt)
{
runAct->AddE(EnergyAbs);
}
```

UserRunAction.hh

```
#ifndef ExN03RunAction_h
#define ExN03RunAction_h 1

#include "G4UserRunAction.hh"
#include "globals.hh"
class G4Run;
class ExN03RunAction : public G4UserRunAction
{
  public:
    ExN03RunAction();
   ~ExN03RunAction();

  public:
    void BeginOfRunAction(const G4Run*);
    void   EndOfRunAction(const G4Run*);
    void AddE(G4double de) {AEnergyAbs +=de;};
private:
    G4double AEnergy;
};
#endif
```

UserRunAction.cc

```
#include "ExN03RunAction.hh"
#include "G4Run.hh"
#include "G4UnitsTable.hh"
ExN03RunAction::ExN03RunAction()
{}
ExN03RunAction::~ExN03RunAction()
{}
void ExN03RunAction::BeginOfRunAction(const G4Run* aRun)
{
G4cout << "### Run " << aRun->GetRunID() << " start." << G4endl;
AEnergy = 0.;
}

void ExN03RunAction::EndOfRunAction(const G4Run* aRun)
{
G4int NbOfEvents = aRun->GetNumberOfEvent();
G4cout << "### Event " << NbOfEvents << " end." << G4endl;
 G4cout<< “the energy deposited in volume A is"
<<G4BestUnit(AEnergy,“Energy")<< G4endl;
}
```