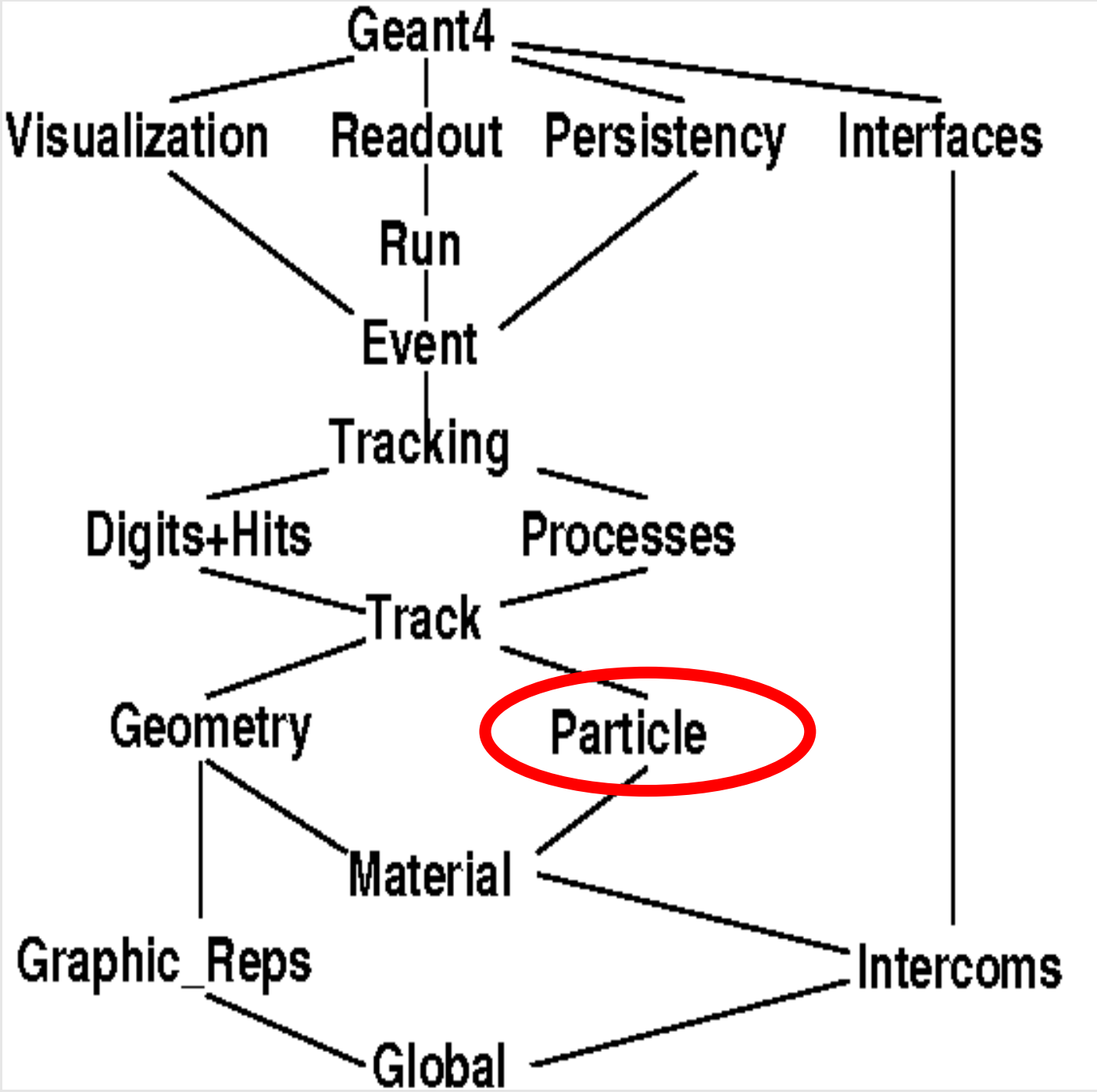
Описание элементарных частиц


Geant4
Visualization
Readout
Persistency
Interfaces
Run
Event
Tracking
Digits+Hits
Processes
Track
Geometry
Particle
Material
Graphic_Reps
Intercoms
Global

Описание элементарных частиц

Каждая частица — это отдельный класс C++ (кроме ионов)

Каждая частица описывается в три этапа:

- **G4ParticleDefinition** - описание постоянных свойств частицы (масса, заряд, название ...)
- **G4DynamicParticle** – описание свойств, изменяющихся при взаимодействии с веществом (энергия, импульс)
- **G4Track** – описание движения частицы в пространстве

Методы G4ParticleDefinition

G4String	GetParticleName()	название
G4double	GetPDGMass()	масса
G4double	GetPDGWidth()	ширина распада
G4double	GetPDGCharge()	заряд
G4double	GetPDGSpin()	спин
G4int	GetPDGiParity()	четность
G4int	GetPDGiConjugation()	зарядовое сопряжение
G4double	GetPDGIsospin()	изоспин
G4double	GetPDGIsospin3()	I_3
G4int	GetPDGiGParity()	G-четность
G4String	GetParticleType()	описание частицы
G4String	GetParticleSubType()	краткое описание частицы
G4int	GetLeptonNumber()	лептонный заряд
G4int	GetBaryonNumber()	барионный заряд
G4int	GetPDGEncoding()	код частицы согласно PDG
G4int	GetAntiPDGEncoding()	код соотв. античастицы

Категории частиц

- **Частицы, участвующие в трекинге**
  - стабильные частицы (протон, электрон, фотон ...)
  - долгоживущие (>10^{-14} с) частицы (пион, мюон ...)
  - короткоживущие частицы, распад которых моделируется в Geant4 (π^0 ...)
  - K-мезоны
  - оптические фотоны
  - geantino
- **Ядра атомов**
  - легкие ядра (дейтрон, альфа-частица, ядро трития)
  - тяжелые ионы
- **Короткоживущие частицы**
  - кварки
  - глюоны
  - мезонные и барионные резонансы

в трекинге не участвуют
появляются только в некоторых
моделях физических процессов

Что такое geantino?

Нейтральная частица, не имеющая физического аналог, не участвующая в физических взаимодействиях, и используемая для отладки транспортировки через объемы детектора.

Существует также заряженное geantino, которое кроме транспортирования через объемы может взаимодействовать с электромагнитным полем.

Генераторы первичной вершины

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"
#include "ExN01DetectorConstruction.hh"
#include "ExN01PhysicsList.hh"
#include "ExN01PrimaryGeneratorAction.hh"
int main()
{
  // construct the default run manager
  G4RunManager* runManager = new G4RunManager;
  // set mandatory initialization classes
  runManager->SetUserInitialization(new ExN01DetectorConstruction);
  runManager->SetUserInitialization(new ExN01PhysicsList);
  // set mandatory user action class
  runManager->SetUserAction(new ExN01PrimaryGeneratorAction);
  // initialize G4 kernel
  runManager->initialize();
  // get the pointer to the UI manager and set verbosities
  G4UImanager* UI = G4UImanager::GetUIpointer();
  UI->ApplyCommand("/run/verbose 1");
  UI->ApplyCommand("/event/verbose 1");
  UI->ApplyCommand("/tracking/verbose 1");
  // start a run
  int numberOfEvent = 3;
  runManager->BeamOn(numberOfEvent);
  // job termination
  delete runManager;
  return 0;
}
```

Класс UserPrimaryGeneratorAction

- Должен быть наследником **G4VUserPrimaryGeneratorAction**
- Должен содержать объект-наследник класса G4VPrimaryGenerator (например G4ParticleGun)
- Должен содержать описание метода generatePrimaries(), в котором происходит вызов метода G4VPrimaryGenerator::generatePrimaryVertex(), создающего первичную вершину
- В одном событии вершина может создаваться при участии нескольких объектов-наследников G4VPrimaryGenerator одновременно

```
#ifndef ExN01PrimaryGeneratorAction_h
#define ExN01PrimaryGeneratorAction_h 1
#include "G4VUserPrimaryGeneratorAction.hh"
class G4ParticleGun;
class G4Event;
class ExN01PrimaryGeneratorAction :
                              public G4VUserPrimaryGeneratorAction
{
  public:
    ExN01PrimaryGeneratorAction();
    ~ExN01PrimaryGeneratorAction();
  public:
    void generatePrimaries(G4Event* anEvent);
  private:
    G4ParticleGun* particleGun;
};
#endif
```

```
ExN01PrimaryGeneratorAction::ExN01PrimaryGeneratorAction()
{
  G4int n_particle = 1;
  particleGun = new G4ParticleGun(n_particle);
  particleGun->SetParticleDefinition(G4Geantino::GeantinoDefinition());
  particleGun->SetParticleEnergy(1.0*GeV);
  particleGun->SetParticlePosition(G4ThreeVector(-2.0*m,0.0*m,0.0*m));
}
ExN01PrimaryGeneratorAction::~ExN01PrimaryGeneratorAction()
{
    delete particleGun;
}
void ExN01PrimaryGeneratorAction::generatePrimaries(G4Event* anEvent)
{
    particleGun->SetParticleMomentumDirection(G4ThreeVector(1.0,0.0,0.0));
    particleGun->generatePrimaryVertex(anEvent);
}
```

G4ParticleGun

- Производит первичную вершину из одной или нескольких частиц с заданными импульсом и начальным положением в пространстве
- Может управляться интерактивно
- Методы:

void SetParticleDefinition(G4ParticleDefinition*)
void SetParticleMomentum(G4ParticleMomentum)
void SetParticleMomentumDirection(G4ThreeVector)
void SetParticleEnergy(G4double)
void SetParticleTime(G4double)
void SetParticlePosition(G4ThreeVector)
void SetParticlePolarization(G4ThreeVector)
void SetNumberOfParticles(G4int)

Интерактивное управление генератором – команды /gun/

- /gun/List показать список частиц
- /gun/particle задать тип частицы
- /gun/direction установить направление вылета
- /gun/energy **установить кинетическую энергию**
- /gun/position установить координаты вершины
- /gun/time установить начальное время
- /gun/polarization задать поляризацию
- /gun/number задать число первичных частиц
- /gun/ion задать свойства иона

Интерфейсы к внешним генераторам

- Кроме G4ParticleGun, в Geant4 существуют интерфейсы к данным в общепринятых форматах
  - **HEPEVT – de facto стандарт выходных данных для генераторов написанных на языке FORTRAN**
  - **HepMC – аналог HEPEVT для генераторов, написанных на C++**

HEPEVT

```
  PARAMETER (NMXHEP=4000)
  COMMON/HEPEVT/NEVHEP,NHEP,ISTHEP(NMXHEP),IDHEP(NMXHEP),
&JMOHEP(2,NMXHEP),JDAHEP(2,NMXHEP),PHEP(5,NMXHEP),VHEP(4,NMXHEP)
  DOUBLE PRECISION PHEP, VHEP
```

Поля записи HEPEVT

- **NMXHEP** – *максимальное число частиц в событии*
- **NEVHEP** – *номер события*
- **NHEP** – *число частиц в событии*
- **ISTHEP(IHEP)** – *статус частицы (1-конечное состояние, 2 – короткоживущая распавшаяся и т.д)*
- **IDHEP(IHEP)** – *PDG код частицы*
- **JMOHEP(1,IHEP)** – *индекс родительской частицы*
- **JDAHEP(1,IHEP)** – *индекс первой дочерней частицы*
- **JDAHEP(2,IHEP)** – *индекс последней дочерней частицы*
- **PHEP(1-5,IHEP)** – *4-импульс и масса частицы*
- **VHEP(1-4,IHEP)** – *координаты вершины и время рождения частицы*

Интерфейс Geant4 для HEPEVT

- Geant4 может читать данные в формате HEPEVT из ASCII файла, используя класс **G4HEPEvtInterface**
- Процедура моделирования с использованием генератора, написанного на языке FORTRAN и записи HEPEVT, выглядит так:
  - **программа-генератор событий на языке FORTRAN дополняется кодом, сохраняющим содержимое COMMON BLOCK HEPEVT в ASCII файл**
  - **Geant4 считывает данные из файла, используя G4HEPEvtInterface**

Пример кода для сохранения записи HEPEVT в ASCII файл

```
************************************************************
      SUBROUTINE HEP2G4
*
* Convert /HEPEVT/ event structure to an ASCII file
* to be fed by G4HEPEvtInterface
*
************************************************************
      PARAMETER (NMXHEP=2000)
      COMMON/HEPEVT/NEVHEP,NHEP,ISTHEP(NMXHEP),IDHEP(NMXHEP),
     >JMOHEP(2,NMXHEP),JDAHEP(2,NMXHEP),PHEP(5,NMXHEP),VHEP(4,NMXHEP)
      DOUBLE PRECISION PHEP,VHEP
*
      WRITE(6,*) NHEP
      DO IHEP=1,NHEP
       WRITE(6,10)
     >  ISTHEP(IHEP),IDHEP(IHEP),JDAHEP(1,IHEP),JDAHEP(2,IHEP),
     >  PHEP(1,IHEP),PHEP(2,IHEP),PHEP(3,IHEP),PHEP(5,IHEP)
10     FORMAT(4I10,4(1X,D15.8))
      ENDDO
*
      RETURN
      END
```

```
#include "G4HEPEvtInterface.hh"

ExN04PrimaryGeneratorAction::ExN04PrimaryGeneratorAction()
{
  HEPEvt = new G4HEPEvtInterface("pythia_event.data");
}

ExN04PrimaryGeneratorAction::~ExN04PrimaryGeneratorAction()
{
  delete HEPEvt;
}

void ExN04PrimaryGeneratorAction::
                                    GeneratePrimaries(G4Event* anEvent)
{
  HEPEvt->SetParticlePosition(G4ThreeVector(0.*cm,0.*cm,0.*cm));
  HEPEvt->GeneratePrimaryVertex(anEvent);
}
```

HepMC

- Набор классов, написанный на C++, и содержащий ту же информацию, что и запись HEPEVT (с некоторыми расширениями)

- Полное описание см. **http://cern.ch/hepmc**

- В Geant4 ввод/вывод с использованием HepMC осуществляется классом G4VHepMCIO

Моделирование отклика детектора

- Любой логический объем в модели можно объявить детектирующим, или «чувствительным». При этом при прохождении частицы через данный объем моделируется срабатывание детектора.
- Детектирующих объемов одновременно может несколько. Обработка срабатываний при этом может происходить по разному.
- Дополнительно можно смоделировать оцифровку сигнала и электронный отклик детектора

Описание детектирующего объема (I)

- Создается класс-наследник класса G4VSensitiveDetector
- Описываются обязательные методы
  - **Initialize**() вызывается в начале каждого события
  - **ProcessHits**() вызывается на каждом шаге в детектирующем объеме. Позволяет получить информацию о характеристиках частицы в данной точке, о взаимодействии с веществом, и смоделировать срабатывание детектора
  - **EndOfEvent**() вызывается в конце события. Позволяет провести отбор срабатываний, и сохранить результаты

Описание детектирующего объема (II)

- Инициализируется объект класса G4SDManager

G4SDManager fSDM = G4SDManager::GetSDMpointer();*

- Объект, описывающий детектирующий объем, регистрируется в менеджере

fSDM->AddNewDetector(G4VSensitiveDetector)*

- Детектирующий объем ассоциируется с логическим объемом

logVolume->SetSensitiveDetector(G4VSensitiveDetector)*

Срабатывание

В Geant4 предусмотрен механизм сбора и сохранения информации о срабатываниях

- **G4VHit** – информация об одном срабатывании Объекты создаются в методе G4VSensitiveDetector::ProcessHits()
- **G4VHitsCollection** – класс-контейнер (коллекция) нескольких срабатываний
- **G4HCofThisEvent** – класс-контейнер разнородных коллекций срабатываний в событии

Пример класса G4VHit

```
class ExN04TrackerHit : public G4VHit
{
  public:

      ExN04TrackerHit();
      ~ExN04TrackerHit();
      void Draw() const;    // обязательная функция
      void Print() const;    // обязательная функция

  private:
      G4double edep;    // ионизационные потери
      G4ThreeVector pos;   // положение срабатывания в пространстве

  public:  // функции доступа к  членам класса
      inline void SetEdep(G4double de)      { edep = de; }
      inline G4double GetEdep() const      { return edep; }
      inline void SetPos(G4ThreeVector xyz)          { pos = xyz; }
      inline G4ThreeVector GetPos() const            { return pos; }
};
```

Инициализация коллекций срабатываний

```
void ExN04TrackerSD::Initialize(G4HCofThisEvent* HCE)
{
  static int HCID = -1;
  trackerCollection = new ExN04TrackerHitsCollection
                    ("SensitiveDetectorName", "collectionName");
  if(HCID<0)
  { HCID = GetCollectionID(0); }
  HCE->AddHitsCollection(HCID,trackerCollection);
}
```

Заполнение информации о срабатывании

```
G4bool ExN04TrackerSD::ProcessHits(G4Step* aStep,
                                   G4TouchableHistory*)
{
  G4double edep = aStep->GetTotalEnergyDeposit();
  if(edep==0.) return false;

  ExN04TrackerHit* newHit = new ExN04TrackerHit();
  newHit->SetEdep( edep );
  newHit->SetPos( aStep->GetPreStepPoint()->GetPosition() );
  trackerCollection->insert( newHit );

  return true;
}
```

Пример доступа к информации о срабатываниях

```
G4SDManager* fSDM = G4SDManager::GetSDMpointer();
G4RunManager* fRM = G4RunManager::GetRunManager();
G4int collectionID =
        fSDM->GetCollectionID("collection_name");
 const G4Event* currentEvent =
        fRM->GetCurrentEvent();
G4HCofThisEvent* HCofEvent =
        currentEvent->GetHCofThisEvent();
MyHitsCollection* myCollection = (MyHitsCollection*)
  (HC0fEvent->GetHC(collectionID));
```

Оцифровка сигнала

- Физические величины, полученные при срабатывании детектора, могут быть преобразованы в электронный сигнал, аналогичный непосредственно регистрируемому системами сбора данных
- Позволяет
  - ***моделировать работу АЦП и время-цифровых преобразователей***
  - ***моделировать схему сбора данных***
  - ***моделировать триггер***
  - ***моделировать наложение событий (pile-up)***
  - ***производить данные в формате, аналогичном формату электроники считывания***

- Каждому модулю электроники соответствует объект-наследник класса G4VDigitizerModule, определяемого пользователем
  - обязательный метод **Digitize**()
- Оцифрованный сигнал хранится в объекте-наследнике класса G4VDigi, определяемого пользователем
- Отдельные оцифрованные сигналы могут объединяться в коллекции (G4VDigiCollection)
- Управление оцифровкой осуществляется объектом класса G4DigiManager

Работа с модулями оцифровки сигнала

Инициализация

```
G4DigiManager * fDM = G4DigiManager::GetDMpointer();
MyDigitizer * myDM = new
   MyDigitizer("/myDet/myCal/myEMdigiMod" );
fDM->AddNewModule(myDM);
```

Доступ к объекту и оцифровка (конструирование объектов-наследников G4VDigi)

```
MyDigitizer * myDM =
     fDM->FindDigitizerModule( "/myDet/myTDCdigiMod" );
 myDM->Digitize();
```

Сохранение результатов моделирования

- **Geant4 не имеет встроенных средств сохранения результатов на диск.**
- Пользователь может использовать любые способы сохранения результатов моделирования, используя методы Geant4 для доступа к этим результатам
  - ASCII файл
  - ROOT
  - JAS
  - HBOOK
  - .....

Использование ROOT
Простой случай

В простых случаях – использование методов ROOT, например, при обработке срабатываний в детектирующем объеме

- Инициализация дерева и/или гистограмм в конструкторе
- Заполнение дерева и/или гистограмм в методах ProcessHits() и/или EndOfEvent()
- Сохранение в ROOT файл в деструкторе

Использование ROOT
Более сложный случай (I)

В сложных случаях – создание синглетного (объект которого существует в программе в единственном экземпляре) класса MyROOTManager:

```
class MyROOTManager
{
public:
static MyROOTManager* GetPointer()
{
    if (theInstance == 0) theInstance = new  MyROOTManager();
    return theInstance;
}
private:
MyROOTManager(){}; // создание объекта только методом GetPointer()
static MyROOTManager* theInstance;
};

MyROOTManager::theInstance = 0;
```

Использование ROOT
Более сложный случай (II)

- Класс MyROOTManager содержит все деревья и гистограммы, обеспечивает заполнение их и запись в файл
- Указатель на объект MyROOTManager в любом месте программы получется вызовом

```
MyROOTManager* fRM = MyROOTManager::GetPointer();
```

- Дальнейшая работа (заполнение гистограмм, запись на диск и т.д.) осуществляется по указателю fRM

Описание электрического и магнитного полей

- Описание электрического и магнитного полей задается при описании геометрии детектора
- Моделирование движения частицы в поле осуществляется численным решением уравнения движения с учетом величины поля в данной точке
- Geant4 позволяет описывать как простые (однородные) поля, так и сложные поля, в том числе задаваемые экспериментально измеренными картами напряженности поля
- Поля могут быть как стационарными, так и изменяющимися во времени

Описание однородного магнитного поля

- Создается объект класса G4UniformMagneticField

```
G4UniformMagField* magField  = new
G4UniformMagField(G4ThreeVector(0.,0.,fieldValue=1*tesla));
```

- Этот объект передается объекту G4FieldManager, которой управляет движением частицы в поле

```
G4FieldManager* fieldMgr
 = G4TransportationManager::GetTransportationManager()->
GetFieldManager();
fieldMgr->SetDetectorField(magField);
```

- Создается объект, рассчитывающий траекторию движения

```
fieldMgr->CreateChordFinder(magField);
```

Описание однородного электрического поля

- Создается объект класса G4UniformElectricField
- Создается объект класса G4EqMagElectricField, задающий уравнение движения в поле
- Создается объект, численно решающий уравнение движения (например G4ClassicalRK4, для решения методом Рунге-Кутта 4-го порядка)
- Аналогично случаю магнитного поля, объект G4UniformElectricField передается объекту G4FieldManager
- Создается объект класса G4MagInt_Driver, применяющий определенный выше метод решения уравнения движения к заданному полю.
- Создается объект, рассчитывающий траекторию движения, и передается в G4FieldManager

```
   G4ElectricField*             fEMfield;
   G4EqMagElectricField*    fEquation;
   G4MagIntegratorStepper* fStepper;
   G4FieldManager*            fFieldMgr;
   G4double                       fMinStep ;
   G4ChordFinder*             fChordFinder ;

   fEMfield = new G4UniformElectricField(G4ThreeVector(0.0,100000.0*kilovolt/cm,0.0)
);

   // уравнение движения в поле
   fEquation = new G4EqMagElectricField(fEMfield);
   G4int nvar = 8; // число переменных

   fStepper = new G4ClassicalRK4( fEquation, nvar );

   fFieldManager= G4TransportationManager::GetTransportationManager()->
       GetFieldManager();

   fFieldManager->SetDetectorField(fEMfield );
   fMinStep     = 0.010*mm ;  // минимальный шаг 10 микрон
   fIntgrDriver = new G4MagInt_Driver(fMinStep,
                             fStepper,  fStepper->GetNumberOfVariables() );

   fChordFinder = new G4ChordFinder(fIntgrDriver);
   fFieldManager->SetChordFinder( fChordFinder );
```

Описание полей в определенном логическом объеме

- Способом, описанным выше, поле задается во всей установке одновременно.
- Однако поле может быть также задано только в определенном логическом объеме, включая или исключая все его дочерние объемы. Для этого создается собственный FieldManager, и передается логическому объему:

```
G4bool includeDaugthers = true;
logicVolumeWithField -> SetFieldManager(MyField ->
                        GetLocalFieldManager(), includeDaugthers);
```

- Подробнее в примере *examples/extended/field/field03*

Указание точности вычислений

Точность вычислений определяется тремя параметрами:

- максимальной длиной шага, на котором заново вычисляется траектория движения
- минимально допустимой относительной ошибкой решения уравнения движения
- максимально допустимой относительной ошибкой решения уравнения движения

```
G4FieldManager * globalFieldManager;

G4TransportationManager *transportMgr=
    G4TransportationManager::GetTransportationManager();

globalFieldManager = transportMgr->GetFieldManager();
                    // Relative accuracy values:
G4double minEps= 1.0e-5;  //   Minimum error value for smallest steps
G4double maxEps= 1.0e-4;  //   Maximum error value for largest steps

globalFieldManager->SetMinimumEpsilonStep( minEps );
globalFieldManager->SetMaximumEpsilonStep( maxEps );
globalFieldManager->SetDeltaOneStep( 0.5e-3 * mm );  // 0.5 um
```

Описание сложных полей

- Необходимо создать свой класс – наследник класса G4Field (в частных случаях только магнитного или электрического полей – соответственно G4MagneticField или G4ElectricField)
- Описать в нем метод

```
void  GetFieldValue( const double Point[3], double *pField );
```

аргументы: Point[3] – координаты точки в пространстве

pField - возвращаемый массив параметров поля в данной точке

- В случае, если поле отличается от простого электрического или магнитного, надо дополнительно описать класс - наследник класса G4Mag_EqRh, в котором задать уравнение движения частицы в поле