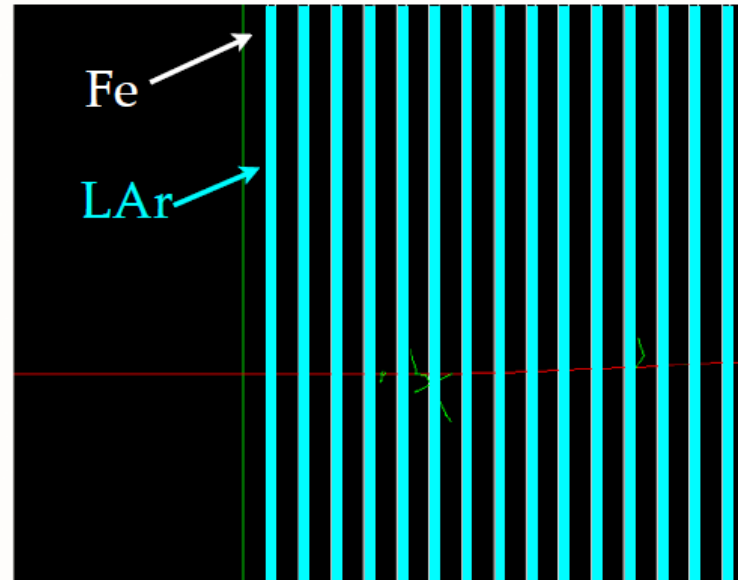


Детектирующие объёмы

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

мюон, 2 ГэВ



Детектирующие объёмы

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 сохранить отклик детектора.
- **EndOfEvent()** вызывается в конце события.
Позволяет провести отбор сохранённых откликов детектора и сохранить результаты.

Protected Member Functions

virtual G4bool	ProcessHits (G4Step *aStep, G4TouchableHistory *ROhist)=0
virtual G4int	GetCollectionID (G4int i)

G4VSensitiveDetector Class Reference

Public Member Functions

	G4VSensitiveDetector (G4String name)
	G4VSensitiveDetector (const G4VSensitiveDetector &right)
virtual	~G4VSensitiveDetector ()
const G4VSensitiveDetector &	operator= (const G4VSensitiveDetector &right)
G4int	operator== (const G4VSensitiveDetector &right) const
G4int	operator!= (const G4VSensitiveDetector &right) const
virtual void	Initialize (G4HCofThisEvent *)
virtual void	EndOfEvent (G4HCofThisEvent *)
virtual void	clear ()
virtual void	DrawAll ()
virtual void	PrintAll ()
G4bool	Hit (G4Step *aStep)
void	SetROgeometry (G4VReadOutGeometry *value)
void	SetFilter (G4VSDFilter *value)
G4int	GetNumberOfCollections () const
G4String	GetCollectionName (G4int id) const
void	SetVerboseLevel (G4int vl)
void	Activate (G4bool activeFlag)
G4bool	isActive () const
G4String	GetName () const
G4String	GetPathName () const
G4String	GetFullPathName () const
G4VReadOutGeometry *	GetROgeometry () const
G4VSDFilter *	GetFilter () const

Protected Attributes

G4CollectionNameVector	collectionName
G4String	SensitiveDetectorName
G4String	thePathName
G4String	fullPathName
G4int	verboseLevel
G4bool	active
G4VReadOutGeometry *	ROgeometry
G4VSDFilter *	filter

Срабатывания или отклики (hits) детекторов в Geant4

Хит в Geant4 – «отпечаток» физического взаимодействия частицы на данном шаге трека в детектирующем (чувствительном) объёме детектора.

«отпечаток» является информацией, которую можно сохранить на данном временном шаге:

- геометрическое положение и время данного шага;
- импульс или энергия частицы на данном шаге;
- потери энергии частицы на данном шаге;
- информация о геометрии детектора, в котором находится частица.

В Geant4 предусмотрен механизм сбора и сохранения информации о срабатываниях в детекторе.

Хиты можно сохранять только для взаимодействий в чувствительном объёме.

Хиты хранятся в контейнере.

Хиты доступны во всех компонентах проекта.

Каждый Хит идентифицируется через персональный идентификатор (номер)

Срабатывания или отклики (hits) детекторов в Geant4

Для трековых детекторов обычно сохраняются хиты для каждого шага трека заряженной частицы:

- пространственное положение
- время
- энергия
- потери энергии
- номер трека

Для калориметра обычно сохраняются хиты для каждой ячейки/элемента :

- суммарное энергосодержание в ячейке в данном событии
- номер ячейки

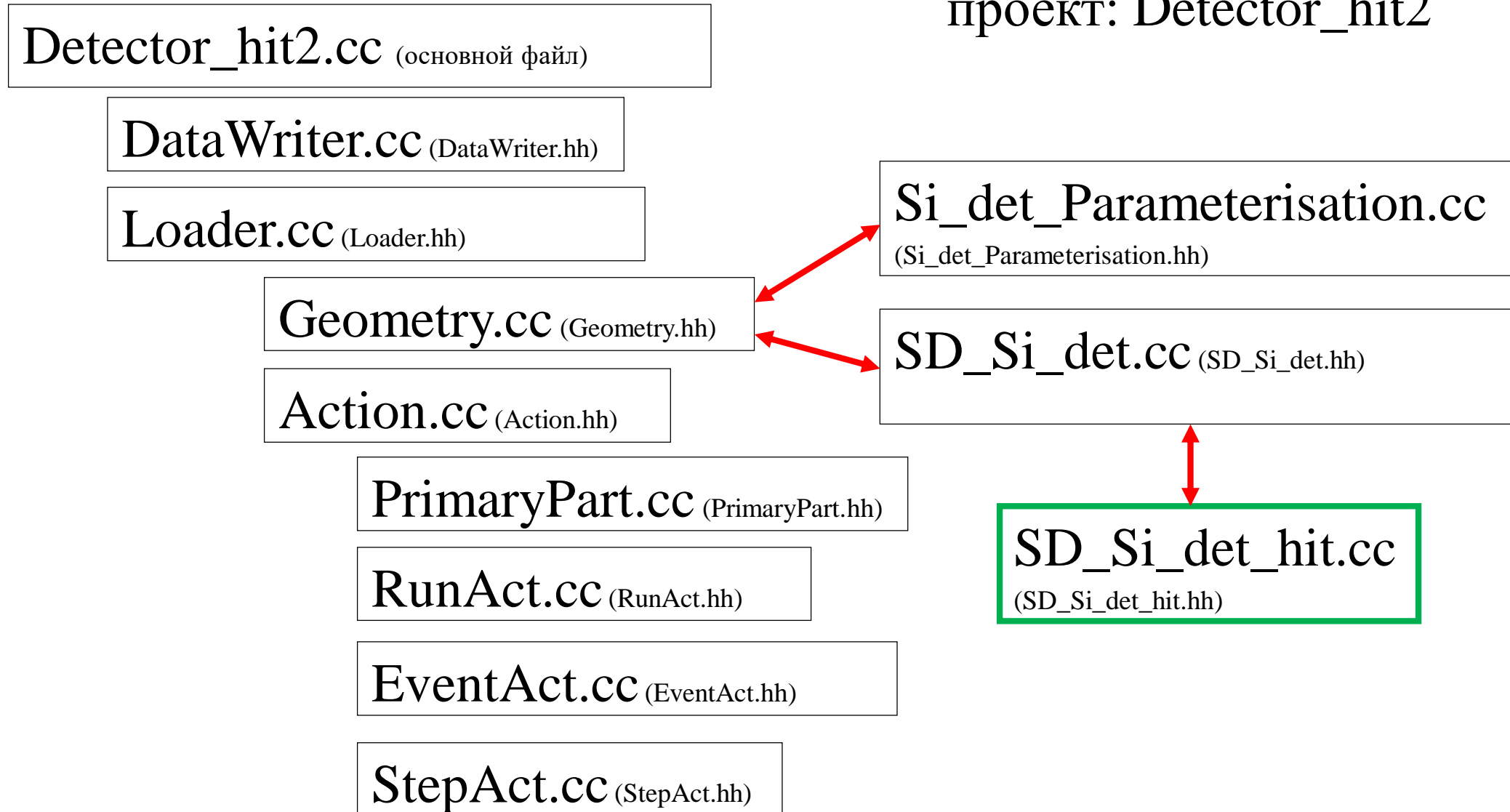
Информация об единичном отклике на текущем шаге описывается в классе –наследнике абстрактного класса **G4VHit**, объекты которого создаются в методе `ProcessHits()` класса чувствительного объёма.

Контейнер (коллекция) для хранения нескольких откликов задаётся с использованием шаблона класса-контейнера **G4VHitsCollection**.

Разные коллекции срабатываний для данного события хранятся в классе-контейнере **G4HCofThisEvent**.

Срабатывания или отклики (hits) детекторов в Geant4

проект: Detector_hit2



```

class SD_Si_det_hit : public G4VHit
{
public:
    SD_Si_det_hit(); ~SD_Si_det_hit();
    SD_Si_det_hit(const SD_Si_det_hit&);

    const SD_Si_det_hit& operator=(const SD_Si_det_hit&);
    G4int operator==(const SD_Si_det_hit&) const;
    inline void* operator new(size_t);
    inline void operator delete(void *aHit);

    void Draw();
    void Print();

    void SetEdep(const double e) {this->eDep_hit=e;}
    G4double GetEdep() const {return eDep_hit;}
    void SetLayerNumber(const int c) {this->layerNumber_hit=c;}
    G4int GetLayerNumber() const {return layerNumber_hit;}

private:
    G4int layerNumber_hit;
    G4double eDep_hit;
};

```

← Наследование класса G4VHit

Конструктор копий

← Перегрузка оператора =

← Перегрузка оператора ==

← Перегрузка оператора new

← Перегрузка оператора delete

← Виртуальные методы класса G4VHit

← Значение энергосвечения для данного срабатывания

← Номер копии детектора для данного срабатывания


← Переменные для данного срабатывания

Срабатывания или отклики (hits) детекторов в Geant4

Sd_Si_det_hit.hh

```
typedef G4THitsCollection<SD_Si_det_hit> SD_Si_det_hitCollection;
```

Создание коллекции хитов SD_Si_det_hitCollection для записи откликов, заданных в классе SD_Si_det_hit



```
extern G4Allocator<SD_Si_det_hit> hitAllocatorSD;
```

```
inline void* SD_Si_det_hit::operator new(size_t) ← Размещение объекта срабатываний в памяти
```

```
{  
    void* aHit;  
    aHit = (void*) hitAllocatorSD.MallocSingle();  
    return aHit;  
}
```

```
inline void SD_Si_det_hit::operator delete(void* aHit) ← Удаление объекта срабатываний из памяти
```

```
{  
    hitAllocatorSD.FreeSingle((SD_Si_det_hit*) aHit);  
}
```

```
#include "SD_Si_det_hit.hh"
```

```
class SD_Si_det : public G4VSensitiveDetector
```

```
{
```

```
public:
```

```
SD_Si_det(G4String SDname);
```

```
~SD_Si_det();
```

```
void Initialize(G4HCofThisEvent* HCE);
```

```
G4bool ProcessHits(G4Step* astep, G4TouchableHistory*);
```

```
void EndOfEvent(G4HCofThisEvent* HCE);
```

```
G4double GetSumE(G4int i) const {return SumE[i];}
```

```
void AddSumE(double e, G4int i) {SumE[i]+=e;}
```

```
private:
```

```
std::ofstream hit_SD_Si_det[10];
```

```
G4double SumE[10];
```

```
SD_Si_det_hitCollection *hitCollection;
```

```
G4int collectionID;
```

```
};
```

← Инициализация коллекции хитов

← Заполнение хитов и запись в коллекцию

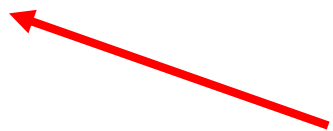
← Запись информации в конце события

← Коллекция хитов — член класса
чувствительных объёмов

Срабатывания или отклики (hits) детекторов в Geant4

Sd_Si_det.cc

```
SD_Si_det :: SD_Si_det (G4String SDname) : G4VSensitiveDetector(SDname)
{
  collectionName.insert("SD_Si_det_hitCollection");
  G4String filename[10];
  char str[2];
  G4int i;
  for (i=0;i<10;i++)
  {
    filename[i] ="hit_Si_det_";
    sprintf(str,"%d",i+1);
    filename[i]+=str;
    filename[i]+=".dat";
    hit_SD_Si_det[i].open(filename[i],std::fstream::out);
  }
  for (i=0; i<10; i++) {SumE[i]=0.;}
}
```



Имя коллекции хитов заносится в вектор имён collectionName в конструкторе класса чувствительных объёмов Sd_Si_det.
collectionName – protected член класса G4VSensitiveDetector

Срабатывания или отклики (hits) детекторов в Geant4

Sd_Si_det.cc

Инициализация коллекции хитов

```
void SD_Si_det :: Initialize(G4HCofThisEvent* HCE)
{
    hitCollection = new SD_Si_det_hitCollection(GetName(), collectionName[0]);
    static G4int HCID=-1;
    if (HCID<0) HCID = GetCollectionID(0);
    HCE->AddHitsCollection(HCID, hitCollection);
}
```

Создание коллекции хитов

Первый элемент [0] в векторе имён collectionName

Возвращает имя коллекции хитов: SD_Si_det_hitCollection

Получение ID для коллекции.

Операция выполняется не для каждого события, только один раз.

Регистрация объекта hitCollection коллекции хитов в общей коллекции хитов HCE для данного события

Срабатывания или отклики (hits) детекторов в Geant4

Sd_Si_det.cc

```
G4bool SD_Si_det :: ProcessHits(G4Step* step, G4TouchableHistory*)
{
    .....
    SD_Si_det_hit *aHit = new SD_Si_det_hit();
    aHit->SetEdep(step->GetTotalEnergyDeposit());
    aHit->SetLayerNumber(step->GetPreStepPoint()->GetTouchableHandle()->GetVolume(0)->GetCopyNo());
    hitCollection->insert(aHit);
    .....
}
```

Создание объекта с хитом

Добавление хита в коллекцию

Сохранение информации в переменных хита

Срабатывания или отклики (hits) детекторов в Geant4

EventAct.cc (Запись результатов)

```
void EventAct::EndOfEventAction(const G4Event *EVE)
```

```
{
```

```
.....
```

```
SD_Si_det_hitCollection *THC = NULL;
```

```
G4HCofThisEvent *HCE = EVE->GetHCofThisEvent();
```

```
if (HCE)
```

```
{
```

```
    THC = (SD_Si_det_hitCollection*)(HCE->GetHC(0));
```

```
}
```

```
if (THC)
```

```
{
```

```
    int n_hit=THC->entries();
```

```
    for (int i=0; i<n_hit; i++)
```

```
    {
```

```
        SD_Si_det_hit *hit = (*THC)[i];
```

```
        G4cout<<"collection "<<hit->GetEdep()<<" "<<hit->GetLayerNumber()<<G4endl;
```

```
    }
```

```
}
```

```
}
```

Доступ к коллекции хитов для данного события

Доступ к коллекции хитов с ID=0

Извлечение хита из коллекции

Вывод на экран

Срабатывания или отклики (hits) детекторов в Geant4



Задача 1 (Резерфорд).

Доступ к информации на шаге вылета ядра гелия из золота.

```
~::UserSteppingAction(const G4Step* step)
{
    G4StepPoint* point1 = step->GetPreStepPoint();
    G4StepPoint* point2 = step->GetPostStepPoint();
    .....
    if (strcmp("G4_Au",step->GetPreStepPoint()->GetMaterial()->GetName())==0)
    {
        if (strcmp("alpha",step->GetTrack()->GetDynamicParticle()->GetDefinition()->GetParticleName())==0)
        {
            if (step->IsLastStepInVolume())
            {
                this->v1=point1->GetPosition();
                this->v2=point2->GetPosition();
            }
        }
    }
    .....
}
```

Задача 1 (Резерфорд).

Доступ к информации на шаге влёта ядра гелия в детектор из кремния.

```
~::UserSteppingAction(const G4Step* step)
{
    .....

    if (strcmp("G4_Si",step->GetPreStepPoint()->GetMaterial()->GetName())==0)
    {
        if (strcmp("alpha",step->GetTrack()->GetDynamicParticle()->GetDefinition()->GetParticleName())==0)
        {
            if (step->IsFirstStepInVolume())
            {
                vect1=point1->GetPosition();
                vect2=point2->GetPosition();
            }
        }
    }
}
```

Генераторы случайных чисел в Geant4

- Реализация моделирования Monte Carlo основывается на применении генераторов случайных чисел для имитации вероятностной природы физических процессов.
- Генератор случайных чисел — это «приблизительно» случайный генератор, использующий детерминированный алгоритм с предсказуемыми результатами.
- Существуют различные алгоритмы генерации чисел, отличающиеся скоростью работы и качеством генерации «случайности».
- Генераторы случайных чисел принято называть “engine” (движок).
- Каждый генератор случайных чисел в качестве начальных данных или начальной точки генерации использует seed (зерно): целое число или набор целых чисел.
- Для фиксированного зерна данный движок всегда генерирует одинаковую последовательность случайных чисел.

Генераторы случайных чисел в Geant4

- Также генератор случайных чисел характеризуется состоянием или статусом.
 - После остановки генерации последовательности из N случайных чисел движок находится в некотором состоянии, которое может быть сохранено (в файле).
 - При загрузке этого состояния в качестве начального при следующем запуске движка начнётся продолжение генерации последовательности случайных чисел, такой же, как и в случае без прерывания.
- Движок генерирует последовательность случайных чисел в интервале $]0,1[$.
Другие распределения могут быть получены из равномерного с применением соответствующих преобразований.
- Geant4 обеспечивает возможности выбора движка, установки значения зерна, сохранения или перегрузки статуса.

Генераторы случайных чисел в Geant4

Контроль состояния движка необходим для:

1. Начальной установки значений зерна/зёрен.

Запуск 10^7 событий на многопроцессорном компьютере можно разбить на 10 статистически независимых запусков для 10^6 событий с различными значениями начальных зёрен.

В результате обеспечивается ускорение расчёта в 10 раз.

2. Сохранение статуса движка.

При аварийном завершении программы необходимо определить причину сбоя.

Если сбой произошёл при моделировании 1000000-ого события нет необходимости моделировать предыдущие 999999 при повторном запуске.

При запуске программы с использованием состояния движка для 999999-го события можно сразу запустить моделирование сбойного события и детально исследовать его.

Сохранение состояния движка для каждого события требует значительных временных ресурсов, данную процедуру **НЕ СЛЕДУЕТ** использовать всегда или по умолчанию.

Выбор движка определяется соотношением между скоростью расчёта и необходимым качеством случайности при генерации ряда случайных чисел.

Генераторы случайных чисел в Geant4

Класс HepRandom – модуль CLHEP (Class Library for High Energy Physics)

Генератор случайных чисел – “engine”

Класс HepRandom - абстрактный класс с интерфейсом для каждого генератора случайных чисел и различных типов распределений случайных чисел.

CLHEP::HepRandom Class Reference

Public Member Functions

	HepRandom ()
	HepRandom (long seed)
	HepRandom (HepRandomEngine &algorithm)
	HepRandom (HepRandomEngine *algorithm)
virtual	~HepRandom ()
double	flat ()
	void flatArray (const int size, double *vect)
double	flat (HepRandomEngine *theNewEngine)
	void flatArray (HepRandomEngine *theNewEngine, const int size, double *vect)
virtual double	operator() ()
virtual std::string	name () const
virtual HepRandomEngine &	engine ()
virtual std::ostream &	put (std::ostream &os) const
virtual std::istream &	get (std::istream &is)

Static Public Member Functions

	static void setTheSeed (long seed, int lux=3)
	static long getTheSeed ()
	static void setTheSeeds (const long *seeds, int aux=-1)
	static const long * getTheSeeds ()
	static void getTheTableSeeds (long *seeds, int index)
	static HepRandom * getTheGenerator ()
	static void setTheEngine (HepRandomEngine *theNewEngine)
static HepRandomEngine *	getTheEngine ()
	static void saveEngineStatus (const char filename[]="Config.conf")
	static void restoreEngineStatus (const char filename[]="Config.conf")
	static std::ostream & saveFullState (std::ostream &os)
	static std::istream & restoreFullState (std::istream &is)
	static std::ostream & saveDistState (std::ostream &os)
	static std::istream & restoreDistState (std::istream &is)
	static std::ostream & saveStaticRandomStates (std::ostream &os)
	static std::istream & restoreStaticRandomStates (std::istream &is)
	static void showEngineStatus ()
	static int createInstance ()
	static std::string distributionName ()

Генераторы случайных чисел в Geant4

Класс `HepRandomEngine` - абстрактный класс с интерфейсом для каждого генератора случайных чисел.

Встроенные генераторы случайных чисел:

- `HepJamesRandom`
(используется по умолчанию)
- `DRand48Engine`
(используются системные функции стандартных библиотек C)
- `MixMaxRng`
(генератор по умолчанию при генерации распределений)
- `RanluxEngine`
(используется генератор языка FORTRAN77, задаётся уровень качества генерации в диапазоне от 1 до 5, по умолчанию 3)
- `RanecuEngine`
(используется генератор языка FORTRAN77, начальные зёрна задаются из таблицы по индексу)

CLHEP::HepRandomEngine Class Reference

Public Member Functions

	<code>HepRandomEngine ()</code>
virtual	<code>~HepRandomEngine ()</code>
bool	<code>operator== (const HepRandomEngine &engine)</code>
bool	<code>operator!= (const HepRandomEngine &engine)</code>
virtual double	<code>flat ()=0</code>
virtual void	<code>flatArray (const int size, double *vect)=0</code>
virtual void	<code>setSeed (long seed, int)=0</code>
virtual void	<code>setSeeds (const long *seeds, int)=0</code>
virtual void	<code>saveStatus (const char filename[]="Config.conf") const =0</code>
virtual void	<code>restoreStatus (const char filename[]="Config.conf")=0</code>
virtual void	<code>showStatus () const =0</code>
virtual std::string	<code>name () const =0</code>
virtual std::ostream &	<code>put (std::ostream &os) const</code>
virtual std::istream &	<code>get (std::istream &is)</code>
virtual std::istream &	<code>getState (std::istream &is)</code>
virtual std::vector< unsigned long >	<code>put () const</code>
virtual bool	<code>get (const std::vector< unsigned long > &v)</code>
virtual bool	<code>getState (const std::vector< unsigned long > &v)</code>
long	<code>getSeed () const</code>
const long *	<code>getSeeds () const</code>
virtual	<code>operator double ()</code>
virtual	<code>operator float ()</code>
virtual	<code>operator unsigned int ()</code>

Генераторы случайных чисел в Geant4

Проект Rand **Loader.cc**

```
CLHEP::HepRandom::setTheEngine(new CLHEP::RanecuEngine);  
CLHEP::HepRandom::setTheSeed(time(NULL)+getpid());  
G4Random::showEngineStatus();  
(*ofs_sn) << "seed=" << CLHEP::HepRandom::getTheSeed() <<G4endl;  
(*ofs_sn) << "seeds=" << *CLHEP::HepRandom::getTheSeeds() <<G4endl;
```

Задание генератора

Задание начального зерна

Вывод статуса генератора

Запись начальных зёрен в info.txt

Уровень загрузки проекта

Сохранение состояния движка в файл Config.conf, загрузка состояния движка из файла Config.conf

```
CLHEP::HepRandom::saveEngineStatus();  
CLHEP::HepRandom::restoreEngineStatus();
```

Сохранение состояния движка в файл с динамически генерируемым именем “Loader_идентификатор процесса.rndm”, загрузка состояния движка из файла “Loader_идентификатор процесса.rndm”.

```
std::ostream os2;  
os2<<"Loader_"<<G4Threading::G4GetThreadId()<<".rndm";  
G4Random::saveEngineStatus(os2.str().c_str());  
G4Random::restoreEngineStatus(os2.str().c_str());
```

преобразование строки в C строку с нулевым символом в конце

Генераторы случайных чисел в Geant4

Проект Rand
PrimaryPart.cc

Уровень генерации вершины

Сохранение состояния движка в файл ./random/run0evt0.rndm, загрузка состояния движка из файла ./random/run0evt0.rndm

```
std::string fileName="./random/run0evt0.rndm";  
G4Random::saveEngineStatus(fileName.c_str());  
G4Random::restoreEngineStatus(fileName.c_str());
```

Задание начальной энергии частицы, распределённой по Гаусу со средним значением 500 МэВ и среднеквадратичным отклонением 100 МэВ

```
G4double EnergyProt =G4RandGauss::shoot(500.*MeV,100.*MeV);  
(*f_prim) << "Initial energy="<<EnergyProt<<G4endl;  
GProton->SetParticleEnergy(EnergyProt);
```



Вывод в файл info.txt

Генераторы случайных чисел в Geant4

Классы распределений в HepRandom:

- RandFlat

```
double fnum = RandFlat::shoot();           // fnum  ]0,1[
double fnum = RandFlat::shoot(n);          // fnum  ]0,n[
double fnum = RandFlat::shoot(-m,n);       // fnum  ]-m,n[
    long inum = RandFlat::shootInt(k);      // inum  [0,k[
    long inum = RandFlat::shootInt(-h,k);   // inum  [-h,k[
        int i = RandFlat::shootBit();      // it returns just a bit (0 or 1)
const int size=n;
double vect[size];
RandFlat::shootArray(size,vect); // to fill an array "vect" of n
                                // double flat values
```

- RandExponential

```
double m;
double num = RandExponential::shoot();    // (mean=1)
double num = RandExponential::shoot(m);   // (mean=m)
```

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

- RandGauss

```
double num = RandGauss::shoot();          // (mean=0) (stDev=1)
double num = RandGauss::shoot(m,s);       // (mean=m, stDev=s)
```

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

- RandBreitWigner

```
double num = RandBreitWigner::shoot(m,g); // (mean=m, gamma=g)
double num = RandBreitWigner::shoot(m,g,c); // (mean=m, gamma=g, cut=c)
```

$$f(E) = \frac{k}{(E^2 - M^2)^2 + M^2\Gamma^2}$$

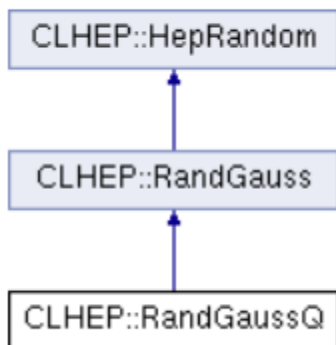
- RandPoisson

```
double m;
long num = RandPoisson::shoot(m); // (mean=m)
```

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}$$

Генераторы случайных чисел в Geant4

Randomize.hh File Reference



Среднее 0, среднеквадратическое отклонение 1.

```
#include <CLHEP/Random/Randomize.h>
#include <CLHEP/Random/RandFlat.h>
#include <CLHEP/Random/RandBit.h>
#include <CLHEP/Random/RandGamma.h>
#include <CLHEP/Random/RandGaussQ.h>
#include <CLHEP/Random/RandPoissonQ.h>
#include <CLHEP/Random/RandExponential.h>
#include <CLHEP/Random/RandGeneral.h>
```

```
#define randomize_h 1
#define G4RandStat CLHEP::HepStat
#define G4RandFlat CLHEP::RandFlat
#define G4RandBit CLHEP::RandBit
#define G4RandGamma CLHEP::RandGamma
#define G4RandGauss CLHEP::RandGaussQ
#define G4RandExponential CLHEP::RandExponential
#define G4RandGeneral CLHEP::RandGeneral
#define G4Random CLHEP::HepRandom
#define G4UniformRand() CLHEP::HepRandom::getTheEngine()->flat()
```

```
G4double EnergyProt = G4RandGauss::shoot(500.*MeV, 100.*MeV);
```