

RunManager

Под названием Geant4 скрываются сотни классов, решающих ту или иную задачу. Однако связь между собой имеет лишь ограниченное число, базовых, а часто абстрактных базовых классов, вместе образующих ядро Geant4. Для инициализации ядра в Geant4 существует специальный класс G4RunManager. Этот класс является управляющим. Он контролирует последовательность выполнения программы, а также управляет циклом событий во время запуска. Кроме того, пользователь может использовать иную версию данного класса — G4MTRunManager, если работает в многопоточном режиме.

Рассмотрим пример инициализации ядра Geant4. Для простоты следует включить в начало нашей программы следующий код:

```
#include <G4RunManager.hh>

int main() {
    G4RunManager runManager;
    return 0;
}
```

В CmakeLists стоит указать путь до Geant4 и связать его с проектом:

```
set(NAME test)
project(${NAME})

set(CMAKE_PREFIX_PATH /home/idalov/geant4/geant4-install)

find_package(Geant4 REQUIRED)
include(${Geant4_USE_FILE})

add_executable(${NAME} main.cpp)
```

Также не стоит забывать, что для корректного выполнения данного кода необходимо указать все пути до файлов данных. В противном случае на экране отобразится сообщение об ошибке

```
----- EEEE ----- G4Exception-START ----- EEEE -----  
*** G4Exception : PART70000  
        issued by : G4NuclideTable  
G4ENSDFSTATEDATA environment variable must be set
```

В случае успешного выполнения программы сообщение будет выглядеть следующим образом:

```
*****  
Geant4 version Name: geant4-10-04-patch-02 [MT] (25-May-2018)  
        Copyright : Geant4 Collaboration  
        References : NIM A 506 (2003), 250-303  
                   : IEEE-TNS 53 (2006), 270-278  
                   : NIM A 835 (2016), 186-225  
        WWW : http://geant4.org/  
*****
```

В большинстве примеров Geant4 используется конструкция следующего вида:

```
#ifdef G4MULTITHREADED  
    auto runManager = new G4MTRunManager;  
#else  
    auto runManager = new G4RunManager;
```

Как можно заметить, за счет значения `G4MULTITHREADED` выбирается, какой конструктор нужно использовать для создания `runManager`. В данном случае это «многопоточный» режим.

Перейдем к обзору основных методов данного класса. Как было сказано ранее, одной из функций `runManager` является связь пользовательских классов с ядром G4. С этой целью предоставлена группа перегрузок метода `SetUserInitialization`.

```
virtual void SetUserInitialization(G4VUserPhysicsList* userPL);  
virtual void SetUserInitialization(G4VUserDetectorConstruction* userDC);  
virtual void SetUserInitialization(G4VUserActionInitialization* userInit);
```

В G4 для корректной работы любого приложения необходимо передать информацию о трех основных столпах моделирования, а именно о том: «Где

происходит моделирование?», «По каким законам происходит моделирование?», «Как происходит моделирование?». На первый вопрос отвечает класс геометрии (подробнее в следующей главе). На второй — класс, содержащий список всех физических процессов, которые могут произойти в рамках данного моделирования, и наконец, на третий вопрос отвечает класс пользовательских действий, включающий информацию о том, какие первичные частицы порождаются.

Таким образом, за счет метода `SetUserInitialization()` нужно передать объекту класса `G4MTRunManager/G4RunManager` указатели на все интересующие объекты.

```
runManager->SetUserInitialization(new Geometry());           //геометрия
runManager->SetUserInitialization(new Shielding);           //процессы
runManager->SetUserInitialization(new Action());             //классы действий
```

После того как вся необходимая информация передана, следует вызвать метод инициализации (`Initialize()`).

```
virtual void Initialize();
```

Как было сказано, `RunManager` является управляющим классом. Это означает, что через него никак не связанные элементы моделирования могут получить доступ к другим частям проекта.

Получить доступ к уже существующему `runManager` можно за счет соответствующего статического Get-метода.

```
static G4RunManager* GetRunManager();
```

Можно воспользоваться любым из Get-методов данного класса, возвращающих указатели на соответствующие части моделирования

```
inline const G4VUserDetectorConstruction* GetUserDetectorConstruction() const
{ return userDetector; }
inline const G4VUserPhysicsList* GetUserPhysicsList() const
{ return physicsList; }
inline const G4VUserActionInitialization* GetUserActionInitialization() const
{ return userActionInitialization; }
```

Отдельного упоминания достоин метод `SetVerboseLevel`.

Данный метод отвечает за уровень подробности, выводимой менеджером информации. К примеру, по умолчанию уровень выводимой информации 0. Если с помощью данного метода установить значение 2, то появится возможность увидеть куда больше информации о том, что происходит внутри

```
inline void SetVerboseLevel(G4int vl)
```

Geant4.

Например, можно заметить, что необходимости удаления большинства создаваемых объектов (предусмотренных ядром G4, разумеется) нет. И все классы удаляются одновременно с разрушением объекта `G4RunManager`.

VisManager

В Geant4 для упрощения процесса отладки, а также с целью большей наглядности процесса моделирования представлен визуальный режим работы.

Если в рамках проекта необходимо визуальное представление, то следует создать объект класса `G4VisExecutive`, наследующий `G4VisManager`.

```
G4VisExecutive (const G4String& verbosityString = "warnings");
```

В зависимости от типа установки Geant4 могут быть доступны различные средства отображения, в рамках данного пособия используется комбинация QT4 и OpenGL. Остальные доступные средства представления, которые могут отличаться в зависимости от операционной системы, представлены в основном руководстве по Geant4, где указана их применимость и особенности.

Для инициализации графической оболочки недостаточно создать объект класса, а, как и в случае с `RunManager`, необходимо вызвать метод инициализации

```
void Initialize ();
```

Однако не стоит считать, что сразу после вызова метода `Initialize()`, появится сцена, содержащая в себе все элементы моделирования.

Стоит отметить, что `G4RunManager` и `G4VisManager` независимы, а, следовательно, их разрушение нужно производить отдельно.

UImanager

С целью повышения гибкости приложений, реализованных с использованием Geant4, был предусмотрен особый режим командной строки, а также группа команд.

Для того, чтобы получить возможность работы с командами, следует воспользоваться статическим методом, возвращающим указатель на объект соответствующего класса.

```
static G4UImanager * GetUImanager();
```

Подробнее использование пользовательских команд будет рассмотрено в соответствующем разделе, однако, стоит отметить, что для включения интерактивного режима необходимо создать объект класса `G4UIExecutive`. В качестве аргументов объект данного класса принимает стандартные аргументы главной функции `main(int argc, char**argv)`

```
G4UIExecutive(G4int argc, char** argv, const G4String& type = "");
```

Для запуска интерактивного режима работы следует воспользоваться методом

```
void SessionStart();
```


По завершению работы не стоит забывать удалять все созданные объекты классов управления:

```
delete ui;  
delete visManager;  
delete runManager;
```

Рабочее окно визуального интерактивного режима

The screenshot displays the Geant4 Qt interface with the following components:

- Top Bar:** Contains standard window controls and a menu bar with "template", "1", and window management icons.
- Scene tree, Help, History:** A sidebar on the left with tabs for "Scene tree", "Help", and "History". The "Scene tree" tab is active, showing a list of objects: "Axes", "Date", and "Logo2D".
- viewer-0 (OpenGLStoredQt):** The main visualization window. It displays a 3D coordinate system with axes labeled "x", "y", and "z". The "x" axis is red and labeled "2 m", the "y" axis is green and labeled "2 m", and the "z" axis is blue and labeled "2 m". A mouse cursor is visible in the center of the viewer.
- Output:** A panel at the bottom right showing the output of the simulation. It includes a "Threads:" dropdown set to "All" and a search icon. The output text reads: "Visualization verbosity changed to quiet (0) /tracking/storeTrajectory 2".
- Session:** A text input field at the bottom left labeled "Session:".

	открыть макрос -файл
---	----------------------

	сохранить текущую сцену в макрос файл
	посмотреть настройки сцены
	сместить сцену
	- открыть окно, чтобы узнать свойства выбранного объекта
	уменьшить масштаб
	увеличить масштаб
	поворачивать сцену
	оставить только ребра объектов, скрывая за невидимыми поверхностями
	показать ребра и поверхности объектов
	показывать все поверхности
	показывать все ребра
	показать сцену в перспективе
	показать сцену в ортогональной проекции
	запустить одно событие

под номером 2 находятся вкладки Scene tree, Help и History

- вкладка Scene tree содержит все графические элементы, отраженные на сцене, включая логотипы, геометрические объекты, оси и т. п.
- вкладка Help содержит все доступные в интерактивном режиме работы команды, включая пользовательские
- вкладка History отражает все вызванные в рамках данной сессии команды

под номером **3** находится сцена, на которой отображается весь процесс моделирования

под номером **4** находится командная строка и история выводимых сообщений.

Если нажать на клавиатуре клавишу Enter, можно вызвать окно захвата видео со сцены.

