



Geant4 physics: particles, processes and physics list

Luciano Pandola

INFN – Laboratori Nazionali del Sud

A lot of material by G.A.P. Cirrone and J. Pipek

XIII International Geant4 School
Prague, July 27th- 31st, 2026

Mandatory (and optional) user classes

At initialization

G4VUserDetectorConstruction

G4VUserPhysicsList

G4VUserActionInitialization

main()
function

At execution

G4VUserPrimaryGeneratorAction

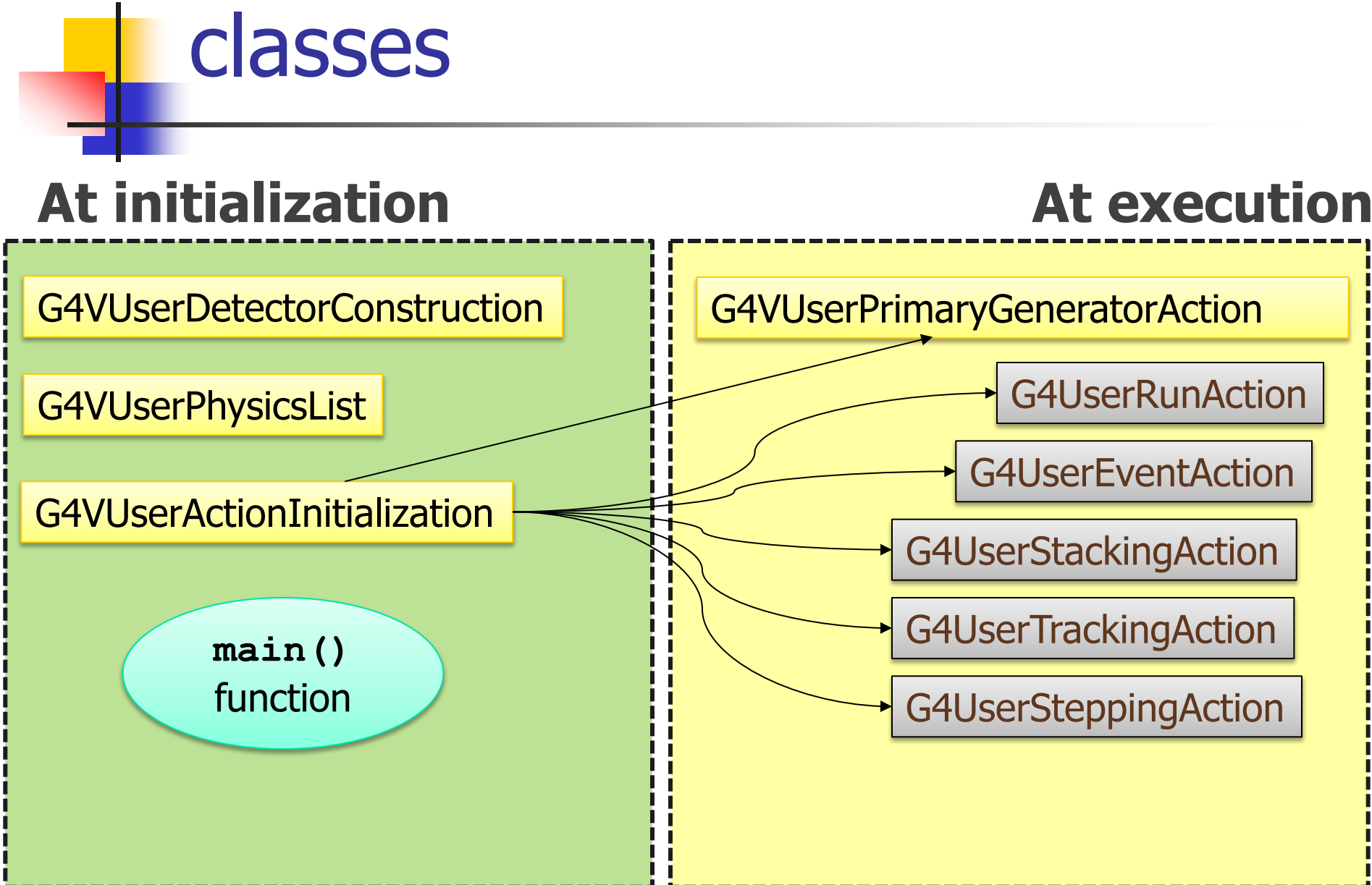
G4UserRunAction

G4UserEventAction

G4UserStackingAction

G4UserTrackingAction

G4UserSteppingAction



Mandatory (and optional) user classes

At initialization

G4VUserDetectorConstruction

G4VUserPhysicsList

G4VUserActionInitialization

main()
function

At execution

G4VUserPrimaryGeneratorAction

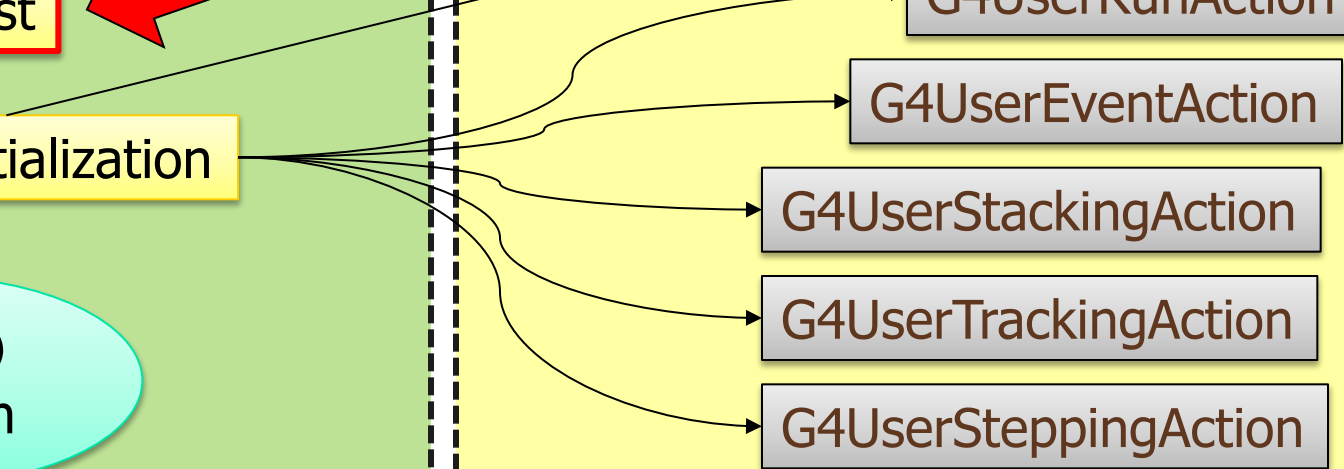
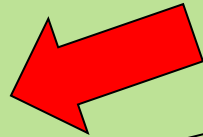
G4UserRunAction

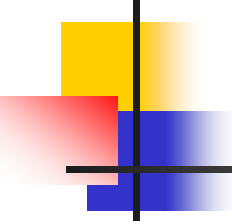
G4UserEventAction

G4UserStackingAction

G4UserTrackingAction

G4UserSteppingAction



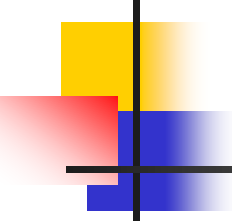


Outlook

- Physics in Geant4 – motivation
- Particles
- Processes
- Physics lists

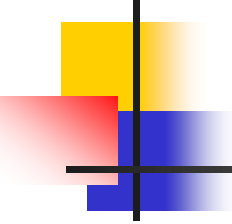
...Part 2:

- Production cuts
- Electromagnetic / hadronic physics



“Shouldn’t there be just one
universal and complete physics
description?”

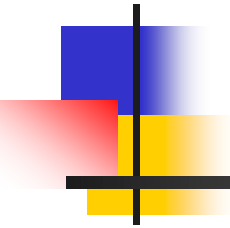
No.



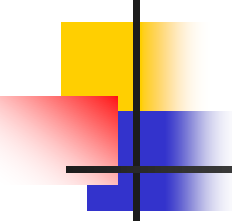
Physics – the challenge

- Huge amount of **different processes** for various purposes (*only a handful relevant*)
- **Competing descriptions** of the **same** physics phenomena (*necessary to choose*)
 - fundamentally different **approaches**
 - balance between **speed** and **precision**
 - different **parameterizations**
- Hypothetical processes & exotic physics

Solution: Atomistic approach with modular **physics lists**

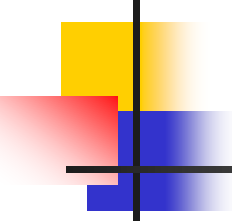


Part I: Particles and Processes



Particles: basic concepts

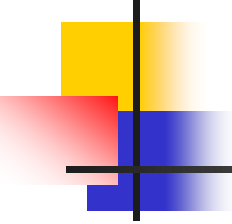
- There are three levels of class to describe particles in Geant4:
- **G4ParticleDefinition**
 - Particle **static properties**: name, mass, spin, PDG number, etc.
- **G4DynamicParticle**
 - Particle **dynamic state**: energy, momentum, polarization, etc.
- **G4Track**
 - Information for tracking **in a detector simulation**: position, step, current volume, track ID, parent ID, etc.



Particles in Geant4

- Particle Data Group (PDG) particles
- Optical photons (different from gammas!)
- Special particles: geantino and charged geantino
 - Only transported in the geometry (no interactions)
 - Charged geantino also feels the EM fields
- Short-lived particles ($\tau < 10^{-14}$ s) are not transported by Geant4 (*decay applied*)
- Light ions (as deuterons, tritons, alphas)
- Heavier ions represented by a single class: G4Ions

Particle name	Class name	Name (in GPS...)	PDG
electron	G4Electron	e-	11
positron	G4Positron	e+	-11
muon +/-	G4MuonPlus G4MuonMinus	mu+ mu-	-13 13
tauon +/-	G4TauPlus G4TauMinus	tau+ tau-	-15 15
electron (anti)neutrino	G4NeutrinoE G4AntiNeutrinoE	nu_e anti_nu_e	12 -12
muon (anti)neutrino	G4NeutrinoMu G4AntiNeutrinoMu	nu_mu anti_nu_mu	14 -14
tau (anti)neutrino	G4NeutrinoTau G4AntiNeutrinoTau	nu_tau anti_nu_tau	16 -16
photon (γ , X)	G4Gamma	gamma	22
photon (optical)	G4OpticalPhoton	opticalphoton	(0)
geantino	G4Geantino	geantino	(0)
charged geantino	G4ChargedGeantino	chargedgeantino	(0)



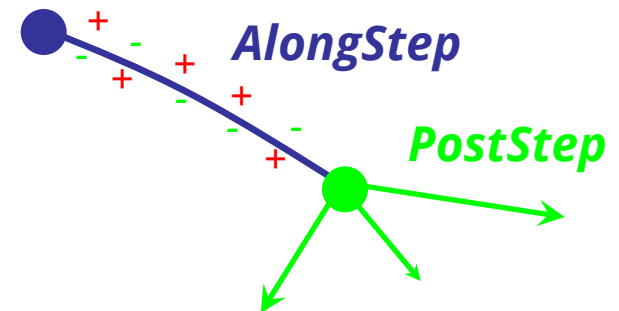
Processes

How do particles interact with materials?

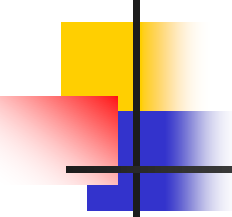
- Responsibilities:
 - decide **when** and **where** an **interaction** occurs
 - GetPhysicalInteractionLength...() → **limit the step**
 - this requires a **cross section**
 - for the transportation process, the **distance** to the **nearest object**
 - **generate** the **final state** of the interaction
 - changes **momentum**, generates **secondaries**, etc.
 - method: DoIt...()
 - this requires **a model of the physics**

The G4VProcess

- Physics processes are derived from the **G4VProcess** base class
- Abstract class defining the **common interface** of all processes in Geant4, used by **all physics processes**
- Three kinds of "actions":
 - **AtRest** actions
 - Decays, e^+ annihilation
 - **AlongStep** actions
 - To describe continuous (inter)actions, occurring along the path of the particle, i.e. **"soft" interactions**
 - **PostStep** actions
 - To describe the point-like (inter)actions, like decay in flight, hadronic interactions, i.e. **"hard" interactions**



A process can implement a combination of them (decay = AtRest + PostStep)



Example processes

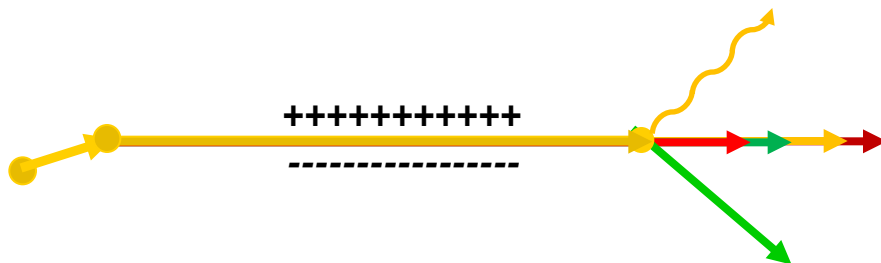
- Discrete process: **Compton Scattering, hadronic inelastic, ...**
 - step determined by cross section, interaction at end of step
 - PostStepGPIL(), PostStepDoIt()
- Continuous process: **Čerenkov effect**
 - photons created along step, roughly proportional to step length
 - AlongStepGPIL(), AlongStepDoIt()
- At rest process: **muon capture at rest**
 - interaction at rest
 - AtRestGPIL(), AtRestDoIt()
- Rest + discrete: **positron annihilation, decay, ...**
 - both in flight and at rest
- Continuous + discrete: **ionization**
 - energy loss is continuous
 - knock-on electrons (δ -ray) are discrete

pure

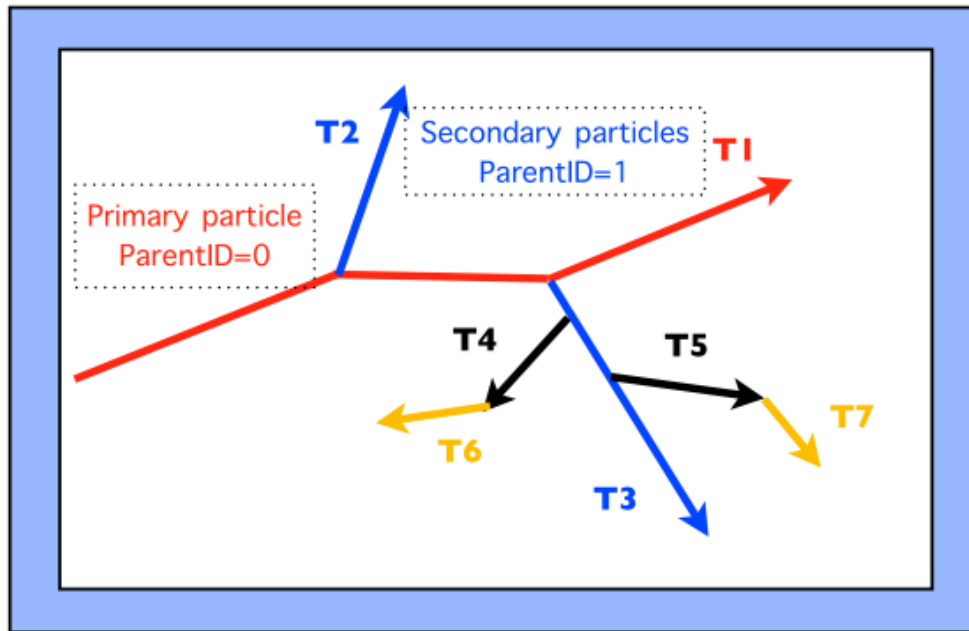
combined

Geant4 transportation in one slide

1. a particle is shot and “transported”
2. all processes associated to the particle **propose** a **geometrical step length**
3. the process proposing the **shortest step** “wins” and the particle is moved to destination (if shorter than “Safety”)
4. **all** processes **along the step** are executed (e.g. ionization)
5. post step phase of the process that limited the step is executed
 - New tracks are “pushed” to the stack
 - Dynamic properties are updated
6. if $E_{\text{kin}}=0$ all at **rest processes** are executed; if particle is stable the track is **killed**
Else
7. new step starts and sequence repeats...



Geant4 way of tracking



- **Force step** at geometry **boundaries**
 - All **AlongStep** processes **co-work**, the **PostStep** **complete** (= only **one** selected)
 - Call **AtRest** actions for particles at rest
-
- Secondaries saved at the top of the stack: tracking order follows 'last in first out' rule:
T1 → **T3** → T5 → **T7** → T4 → **T6** → **T2**

Tracking verbosity

UI command: `/tracking/verbose 1`

Primary γ

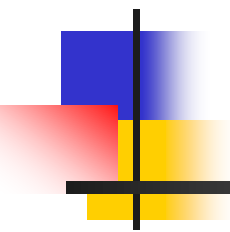
```
*****
* G4Track Information:  Particle = gamma,  Track ID = 1,  Parent ID = 0
*****
```

Step#	X (mm)	Y (mm)	Z (mm)	KinE (MeV)	dE (MeV)	StepLeng	TrackLeng	NextVolume	ProcName
0	47.4	-53	-150	6	0	0	0	Envelope	initStep
1	47.4	-53	-58	0.844	0	92	92	Envelope	compt
2	-46	15.9	5.55	0.47	0	132	224	Envelope	compt
3	-100	6.37	-3.62	0.47	0	55.6	280	World	
Transportation									
4	-120	2.84	-7.02	0.47	0	20.6	301	OutOfWorld	
Transportation									

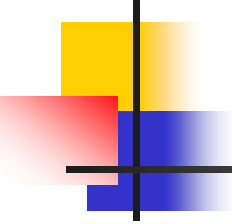
Compton e⁻

```
*****
* G4Track Information:  Particle = e-,  Track ID = 3,  Parent ID = 1
*****
```

Step#	X (mm)	Y (mm)	Z (mm)	KinE (MeV)	dE (MeV)	StepLeng	TrackLeng	NextVolume	ProcName
0	-46	15.9	5.55	0.375	0	0	0	Envelope	initStep
1	-46.1	16.4	5.98	0.0482	0.327	1.16	1.16	Envelope	eIoni
2	-46.1	16.3	5.98	0	0.0482	0.0408	1.2	Envelope	eIoni

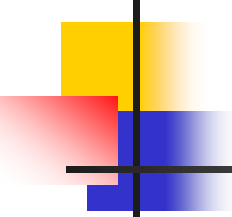


Part II: Physics lists & Co.



A physics list: what it is, what it does

- One instance per application
 - registered to run manager in `main()`
 - inheriting from `G4VUserPhysicsList`
- Responsibilities
 - all particle types (electron, proton, gamma, ...)
 - all processes (photoeffect, bremsstrahlung, ...)
 - all process parameters (...)
 - production cuts (e.g. 1 mm for electrons, ...)



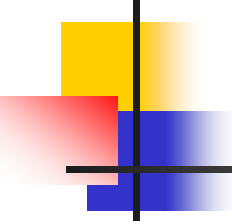
G4VUserPhysicsList

- All **physics lists** **must** derive from this class
 - And then be **registered** to the G4(MT)RunManager
 - **Mandatory** class in Geant4

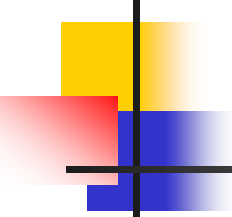
```
class MyPhysicsList: public G4VUserPhysicsList {  
public:  
    MyPhysicsList();  
    ~MyPhysicsList();  
    void ConstructParticle();  
    void ConstructProcess();  
    void SetCuts();  
}
```

- User **must implement** the following (purely virtual) **methods**:
 - `ConstructParticle()`, `ConstructProcess()`
- **Optional Virtual method**:
 - `SetCuts()`

Three ways to get a physics list



- **Manual:** Write your own class, to specify all particles & processes that may occur in the simulation (very flexible, but difficult)
- **Physics constructors:** Combine your physics from pre-defined sets of particles and processes. Still you define your own class – modular physics list (easier)
- **Reference physics lists:** Take one of the pre-defined physics lists. You don't create any class (easy)



Derived class from G4VUserPhysicsList

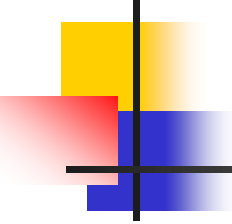
- Implement 3 methods:

```
class MyPhysicsList : public G4VUserPhysicsList {  
public:  
    // ...  
    void ConstructParticle();    // pure virtual  
    void ConstructProcess();    // pure virtual  
    void SetCuts();  
    // ...  
}
```

Advantage: most flexible

Disadvantages:

- most verbose
- most difficult to get right

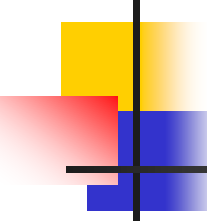


G4VUserPhysicsList: implementation

- `ConstructParticle()`
 - choose the particles you need in your simulation, define **all of them** here
- `ConstructProcess()`
 - for each particle, assign **all the physics processes** relevant to your simulation
- `SetCuts()`

MORE ON THIS LATER

 - set the **range cuts for secondary production** for processes with infrared divergence



```

void StandardPhysics::ConstructParticle()
{
    // We are interested in gamma, electrons and possibly positrons
    G4Electron::ElectronDefinition();
    G4Positron::PositronDefinition();
    G4Gamma::GammaDefinition();
}

void StandardPhysics::ConstructProcess()
{
    // Transportation is necessary
    AddTransportation();

    // Electrons
    G4ProcessManager *elManager = G4Electron::ElectronDefinition()->GetProcessManager();
    elManager->AddProcess(new G4eMultipleScattering, -1, 1, 1);
    elManager->AddProcess(new G4eIonisation, -1, 2, 2);
    elManager->AddProcess(new G4eBremsstrahlung, -1, -1, 3);
    elManager->AddDiscreteProcess(new G4StepLimiter);

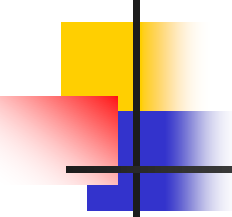
    // Positrons
    G4ProcessManager *posManager = G4Positron::PositronDefinition()->GetProcessManager();
    posManager->AddProcess(new G4eMultipleScattering, -1, 1, 1);
    posManager->AddProcess(new G4eIonisation, -1, 2, 2);
    posManager->AddProcess(new G4eBremsstrahlung, -1, -1, 3);
    posManager->AddProcess(new G4eplusAnnihilation, 0, -1, 4);
    posManager->AddDiscreteProcess(new G4StepLimiter);

    // Gamma
    G4ProcessManager *phManager = G4Gamma::GammaDefinition()->GetProcessManager();
    phManager->AddDiscreteProcess(new G4ComptonScattering);
    phManager->AddDiscreteProcess(new G4PhotoElectricEffect);
    phManager->AddDiscreteProcess(new G4GammaConversion);

    // TODO: Introduce Rayleigh scattering. It has large cross-section than Pair production
}

void StandardPhysics::SetCuts()
{
    // TODO: Create a messenger for this
    defaultCutValue = 0.03 * mm;
    SetCutsWithDefault();
}

```



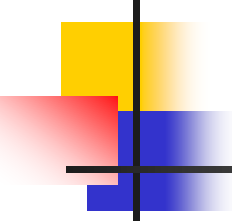
G4VModularPhysicsList

- Similar structure as **G4VUserPhysicsList** (same methods to override – though not necessary):

```
class MyPhysicsList : public G4VModularPhysicsList {  
public:  
    MyPhysicsList();           // define physics constructors  
    void ConstructParticle();   // optional  
    void ConstructProcess();    // optional  
    void SetCuts();             // optional  
}
```

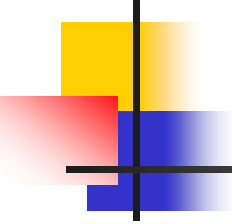
Differences to “manual” way:

- Particles and processes typically handled by **physics constructors** (still customizable)
- **Transportation** automatically included



Physics constructors (1)

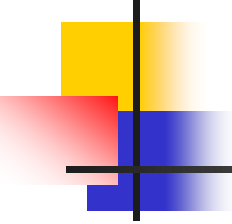
- **"Building blocks"** of a modular physics list
- Inherit from **G4VPhysicsConstructor**
- Defines **ConstructParticle()** and **ConstructProcess()**
 - to be fully imported **in modular list** (behaving in the same way)
- **GetPhysicsType()**
 - enables **switching physics** of the same type, if possible (see next slide)



Physics constructors (2)

- Huge set of **pre-defined ones**
 - **EM**: Standard, Livermore, Penelope
 - **Hadronic inelastic**: QGSP_BIC, FTFP_Bert, ...
 - **Hadronic elastic**: G4HadronElasticPhysics, ...
 - ... (decay, optical physics, EM extras, ...)
- You can implement **your own** (*of course*) by **inheriting** from the **G4VPhysicsConstructor** class

Code: `$G4INSTALL/source/physics_lists/constructors`



How to use physics constructors

Add **physics constructor** in the class **constructor**:

```
MyModularList::MyModularList() {  
    // Hadronic physics  
    RegisterPhysics(new G4HadronElasticPhysics());  
    RegisterPhysics(new G4HadronPhysicsFTFP_BERT_TRV());  
    // EM physics  
    RegisterPhysics(new G4EmStandardPhysics());  
}
```

This **already works** and no further method overriding is necessary 😊

To be continued (if you want to customize)...

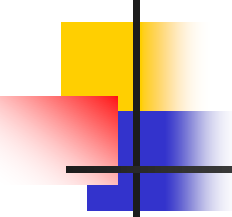
Customizing a G4ModularPhysicsList

- You can override the `CreateParticle()`, `CreateProcess()`, and `SetCuts()` methods:

```
void MyModularList::ConstructProcess() {  
    // Call the default implementation, otherwise you break the behaviour  
    G4VModularPhysicsList::ConstructProcess();  
  
    // Add your customization  
    G4ProcessManager *elManager = G4Electron::Definition()->GetProcessManager();  
    elManager->AddDiscreteProcess(new MyElectronProcess);  
}
```



Don't
forget!

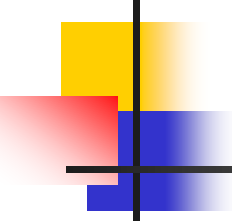


Replace physics constructors

You can **add** or **remove** the physics constructors after the list instance is created:

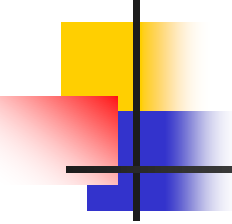
- e.g. in response to **UI command**
- only **before initialization**
- physics of the same type can be **replaced**

```
void MyModularList::SelectAlternativePhysics() {  
    AddPhysics(new G4OpticalPhysics);  
    RemovePhysics(fDecayPhysics);  
    ReplacePhysics(new G4EmLivermorePhysics);  
}
```



Reference physics lists

- Pre-defined ("plug-and-play") physics lists
 - already containing a **complete set** of particles & processes (that work together)
 - **targeted** at specific area of interest (HEP, medical physics, ...)
 - constructed as **modular physics lists**, built on top of **physics constructors**
 - **customizable** (by calling appropriate methods before initialization)



Using a reference physics list

- Super-easy: in the `main()` function, just register an instance of the physics list to the **G4 (MT) RunManager**:

```
#include "QGSP_BERT.hh"

int main() {
    // Run manager
    G4RunManager * runManager = new G4RunManager();
    // ...
    G4VUserPhysicsList* physics = new QGSP_BERT();
    // Here, you can customize the "physics" object
    runManager->SetUserInitialization(physics);
    // ...
}
```

The complete lists of Reference Physics List

`source/physics_lists/lists/include`

FTFP_BERT.hh	FTF_BIC.hh	G4PhysListStamper.hh	QBBC_ABLA.hh	QGSP_FTFP_BERT.hh
FTFP_BERT_ATL.hh	G4GenericPhysicsList.hh	G4RegisterPhysLists.icc	QGSP_BERT.hh	QGSP_INCLXX.hh
FTFP_BERT_HP.hh	G4GenericPhysicsList.icc	INCLXXPhysicsListHelper.hh	QGSP_BERT_HP.hh	QGSP_INCLXX_HP.hh
FTFP_BERT_TRV.hh	G4PhysListFactory.hh	INCLXXPhysicsListHelper.icc	QGSP_BIC.hh	QGS_BIC.hh
FTFP_INCLXX.hh	G4PhysListFactoryAlt.hh	LBE.hh	QGSP_BIC_AllHP.hh	Shielding.hh
FTFP_INCLXX_HP.hh	G4PhysListFactoryMessenger.hh	NuBeam.hh	QGSP_BIC_HP.hh	ShieldingLEND.hh
FTFQGSP_BERT.hh	G4PhysListRegistry.hh	QBBC.hh	QGSP_BIC_HPT.hh	

 Geant4 Homepage

PhysicsListGuide



11.4 (doc Rev11.4)

Search docs

CONTENTS:

Physics List Guide

Reference Physics Lists

[Docs](#) » Guide for Physics Lists

Guide for Physics Lists

Scope of this Manual

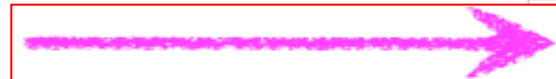
This guide is a description of the physics lists class which is one of the mandatory user classes for a GEANT4 application. For the most part the “reference” physic lists included in the source distribution are described here as well the modularity and electronic options. Some use cases and areas of application are also described.

Contents:

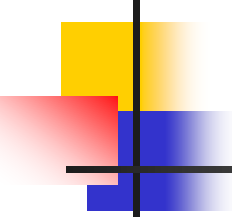
- [Physics List Guide](#)
 - [Bibliography](#)
- [Reference Physics Lists](#)
 - [FTFP_BERT](#)

Where to find information?

<https://www.geant4.org/docs/>

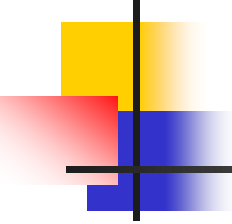


 On this page
Introduction to Geant4
Installation Guide
User guides
For Application Developers
For Toolkit Developers
Physics Reference Manual
Physics List Guide
Examples
Tutorials
Frequently Asked Questions
Geant4 source code



Summary – three kinds of physics lists for Geant4

- Old-style flat physics list
 - You code **what you want**, particle by particle and process by process
 - Very much flexible, but **not really encouraged**
- User-custom modular physics list
 - **Blocks** (constructors) **provided** by Geant4
 - Can register **user-custom** constructors
 - Usually the *optimal compromise* between flexibility and user-friendliness
- Ready-for-the-use Geant4 physics list
 - **Plug and play** (directly registered in the main!)
 - Can still register **extra constructors**



Hands-on session

- Task3
 - Task3a: Particles and processes
 - Task3b: Physics lists
- <http://geant4.lns.infn.it/prague2026/task3>