

Examples

Witold Pokorski, Alberto Ribon
CERN PH/SFT



Learning and Working with Geant4 Examples

- Most of the people, **learn** how to use Geant4 by “playing” with an existing example
 - **Choose** an example as close as possible to your use-case
 - **Look** at the code, to see how things are done
 - **Read** the relevant parts of the “*User's Guide: For Application Developers*” to understand better how things work
 - **Modify** the example to do what you need
- Beside learning: many **real detector-simulation applications** originated from a Geant4 example
 - by adapting the detector description, sensitive detectors, hits, primary source, user actions, and analysis
- Bottom line: **examples are very useful!**

Geant4 Examples

- ***geant4/examples/***
 - ***basic/*** : oriented to **novice users** and covering the **most typical use-cases** of Geant4 applications
 - ***extended/*** : covers many **specific use-cases**;
may require some additional libraries besides of G4
 - ***advanced/*** : **real and complete applications** for different simulation studies;
may require additional third-party products to be built
- There are **README** files in each directory which briefly explain the content of each directory...

Geant4 Basic Examples

- ***geant4/examples/basic/*** :
 - ***B1/***
Simple geometry with a few solids.
Scoring total dose in a selected volume; user action classes.
 - ***B2/***
Simplified tracker geometry with global constant magnetic field.
Scoring within tracker via G4 sensitive detector and hits.
 - ***B3/***
Schematic Positron Emitted Tomography system.
Radioactive source.
Scoring within Crystals via G4 scorers.
 - ***B4/***
Simplified calorimeter with layers of two materials.
Scoring within layers in four ways: (a) via user actions;
(b) via user data object; (c) via hits & sensitive detectors; (d) via scorer
 - ***B5/***
A double-arm spectrometer with wire chambers, hodoscopes and calorimeters with a local constant magnetic field.
Scoring used in wire chambers;
G4 sensitive detector and hits used for hodoscopes and calorimeters.

Geant4 Extended Examples

- ***geant4/examples/extended*** :
 - ***analysis/***
 - ***biasing/*** : examples of event biasing, scoring and reverse-MC
 - ***common/***
 - ***electromagnetic/*** : many, different things...
 - ***errorpropagation/***
 - ***eventgenerator/*** : examples of G4ParticleGun, G4GeneralParticleSource, HepMC, and Pythia
 - ***exoticphysics/*** : examples of monopoles, phonons, ultra-cold neutrons, and channeling
 - ***field/***
 - ***g3tog4/***
 - ***geometry/***
 - ***hadronic/*** : G4PhysListFactory, cross sections, ions, neutron-HP, etc.
 - ***medical/***
 - ***optical/***
 - ***parallel/*** : examples of event-level parallelism
 - ***parameterisations/*** : examples of fast shower parameterisations
 - ***persistency/*** : of geometry (GDML) and simulation output (ROOT I/O)
 - ***polarisation/***
 - ***radioactivedecay/***
 - ***runAndEvent/*** : MC-true, scorers, parallel worlds, regions, readout geometry⁵
 - ***visualization/***

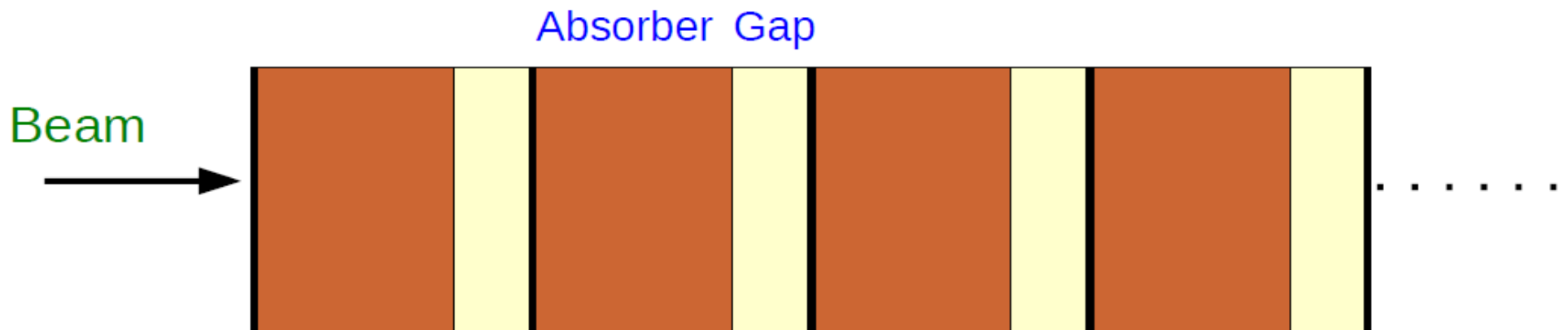
Geant4 Advanced Examples

- ***geant4/examples/advanced*** :

- *air_shower/*
- *amsEcal/*
- *brachytherapy/*
- *ChargeExchangeMC/*
- *composite_calorimeter/*
- *eRosita/*
- *gammaknife/*
- *gammaray_telescope/*
- *hadrontherapy/*
- *human_phantom/*
- *ior_t_therapy/*
- *lAr_calorimeter/*
- *medical_linac/*
- *microbeam/*
- *microelectronics/*
- *nanobeam/*
- *purging_magnet/*
- *radioprotection/*
- *underground_physics/*
- *xray_fluorescence/*
- *xray_telescope/*

A closer look to the basic example B4

- *geant4/examples/basic/B4*
 - a simple sampling calorimeter setup
 - 4 variants of scoring:
 - *B4a/* : user actions
 - *B4b/* : user data object (and user actions)
 - *B4c/* : hits and sensitive detectors
 - *B4d/* : scorer



- The **calorimeter** is a box made of a number of layers
- A **layer** consists of an **absorber plate** and of a **detection gap**
- The layer is **replicated**

Content of *geant4/examples/basic/B4/B4a/*

- **CMakeLists.txt** : to build the example using Cmake (recommended)
- **GNUmakefile** : to build the example with the old GNUmake system (deprecated)
- **exampleB4a.cc** : the main program
- **exampleB4.in** : macro file (there are also others: run1.mac, run2.mac, ... *.mac)
- **include/** : header files (.hh) of the example:
 - B4DetectorConstruction.hh**
 - B4aActionInitialization.hh**
 - B4PrimaryGeneratorAction.hh**
 - B4RunAction.hh**
 - B4aEventAction.hh**
 - B4aSteppingAction.hh**
- **src/** : source files (.cc) of the example:
 - B4DetectorConstruction.cc**
 - B4aActionInitialization.cc**
 - B4PrimaryGeneratorAction.cc**
 - B4RunAction.cc**
 - B4aEventAction.cc**
 - B4aSteppingAction.cc**

A look into a G4 macro file: exampleB4.in

```
# e+ 300MeV
/-gun/particle e+
/-gun/energy 300 MeV
/run/beamOn 1
#
# list the existing physics processes
/process/list
#
# switch off MultipleScattering
/process/inactivate msc
/run/beamOn 1
#
# switch on MultipleScattering
/process/activate msc
#
# change detector parameter
/-gun/particle gamma
/-gun/energy 500 MeV
/run/beamOn 1
```

3 runs, each with a different configuration;
1 event for each run

A look into a G4 main program: exampleB4a.cc

```
...
int main( int argc, char** argv ) {
    ...
    // Build the detector
    B4DetectorConstruction* detConstruction = new B4DetectorConstruction();
    runManager->SetUserInitialization( detConstruction );

    // Choose the physics list
    G4VModularPhysicsList* physicsList = new FTFP_BERT;
    runManager->SetUserInitialization( physicsList );

    // Instantiate the user actions
    B4aActionInitialization* actionInitialization
        = new B4aActionInitialization( detConstruction );
    runManager->SetUserInitialization( actionInitialization );

    // Initialize G4 kernel
    runManager->Initialize()
    ...
}
```

B4DetectorConstruction (1)

```
G4VPhysicalVolume*  
B4DetectorConstruction::Construct() {  
    DefineMaterials();  
    return DefineVolumes();  
}
```

```
void B4DetectorConstruction::DefineMaterials() {  
    // Lead material defined using NIST Manager  
    G4NistManager* nistManager = G4NistManager::Instance();  
    nistManager->FindOrBuildMaterial( "G4_Pb" );  
  
    // Liquid argon material  
    G4double a; // mass of a mole  
    G4double z; // number of protons  
    G4double density;  
    new G4Material( "liquidArgon", z=18., a= 39.95*g/mole, density= 1.390*g/cm3 );  
  
    // Vacuum  
    new G4Material( "Galactic", z=1., a=1.01*g/mole,density= universe_mean_density,  
                  kStateGas, 2.73*kelvin, 3.e-18*pascal );  
  
    // Print materials  
    G4cout << *( G4Material::GetMaterialTable() ) << G4endl;  
}
```

```
int main( int argc, char** argv ) {  
    ...  
    // Build the detector  
    B4DetectorConstruction* detConstruction = new B4DetectorConstruction();  
    runManager->SetUserInitialization( detConstruction );  
  
    // Choose the physics list  
    G4VModularPhysicsList* physicsList = new ETP_BERT;  
    runManager->SetUserInitialization( physicsList );  
    // Instantiate the primary generator and the user actions  
    B4aActionInitialization* actionInitialization  
        = new B4aActionInitialization( detConstruction );  
    runManager->SetUserInitialization( actionInitialization );  
    // Initialize G4 kernel  
    runManager->Initialize()  
    ...  
}
```

B4DetectorConstruction (2)

```
G4VPhysicalVolume* B4DetectorConstruction::DefineVolumes() {  
    ...  
    // --- World ---  
    G4VSolid* worldS = new G4Box( "World",           // its name  
                                worldSizeXY/2, worldSizeXY/2, worldSizeZ/2 ); // its size  
    G4LogicalVolume* worldLV = new G4LogicalVolume( worldS,           // its solid  
                                                    defaultMaterial, // its material  
                                                    "World" );           // its name  
    G4VPhysicalVolume* worldPV = new G4PVPlacement( 0,                // no rotation  
                                                    G4ThreeVector(), // at (0,0,0)  
                                                    worldLV,           // its logical volume  
                                                    "World",           // its name  
                                                    0,                // its mother volume  
                                                    false,           // no boolean operation  
                                                    0,                // copy number  
                                                    fCheckOverlaps );  
  
    // --- Calorimeter ---  
    G4VSolid* calorimeterS = new G4Box( "Calorimeter", calorSizeXY/2, calorSizeXY/2,  
                                       calorThickness/2 );  
    G4LogicalVolume* calorLV = new G4LogicalVolume( calorimeterS, defaultMaterial,  
                                                    "Calorimeter" );  
    new G4PVPlacement( 0, G4ThreeVector(), calorLV, "Calorimeter", worldLV, false, 0,  
                      fcheckOverlaps );  
}
```

CONTINUE...

B4DetectorConstruction (3)

```
// --- Layer ---
G4VSolid* layerS = new G4Box( "Layer", calorSizeXY/2, calorSizeXY/2,
                                layerThickness/2 );
G4LogicalVolume* layerLV = new G4LogicalVolume( layerS, defaultMaterial, "Layer" );
new G4PVReplica( "Layer",           // its name
                layerLV,           // its logical volume
                calorLV,           // its mother
                kZAxis,           // axis of replication
                nofLayers,        // number of replica
                layerThickness ); // width of replica

// --- Absorber ---
G4VSolid* absorberS = new G4Box( "Abso", calorSizeXY/2, calorSizeXY/2,
                                    absoThickness/2 );
G4LogicalVolume* absorberLV = new G4LogicalVolume( absorberS, absorberMaterial,
                                                        "Abso" );
fAbsorberPV = new G4PVPlacement( 0, G4ThreeVector( 0., 0., -gapThickness/2 ),
                                   absorberLV, "Abso", layerLV, false, 0,
                                   fCheckOverlaps );

// --- Gap ---
G4VSolid* gapS = new G4Box( "Gap", calorSizeXY/2, calorSizeXY/2, gapThickness/2 );
G4LogicalVolume* gapLV = new G4LogicalVolume( gapS, gapMaterial, "Gap" );
fGapPV = new G4PVPlacement( 0, G4ThreeVector( 0., 0., absoThickness/2 ), gapLV, "Gap",
                              layerLV, false, 0, fCheckOverlaps );

...
return worldPV;
}
```

B4aActionInitialization

```
void B4aActionInitialization::Build() const {
```

```
    SetUserAction( new B4PrimaryGeneratorAction );
```

```
    SetUserAction( new B4RunAction );
```

```
    B4aEventAction* eventAction = new B4aEventAction;  
    SetUserAction( eventAction );
```

```
    SetUserAction( new B4aSteppingAction( fDetConstruction, eventAction ) );
```

```
}
```

```
int main( int argc, char** argv ) {  
    ...  
    // Build the detector  
    B4DetectorConstruction* detConstruction = new  
    B4DetectorConstruction();  
    runManager->SetUserInitialization( detConstruction );  
    // Choose the physics list  
    G4VModularPhysicsList* physicsList = new FTFP_BERT;  
    runManager->SetUserInitialization( physicsList );  
  
    // Instantiate the primary generator and the user actions  
    B4aActionInitialization* actionInitialization  
    = new B4aActionInitialization( detConstruction );  
    runManager->SetUserInitialization( actionInitialization );  
  
    // Initialize G4 kernel  
    runManager->Initialize()  
    ...  
}
```

In this example, 2 user actions are not used:

- Tracking action
- Stacking action

B4PrimaryGeneratorAction

```
void B4aActionInitialization::Build() const {  
    SetUserAction( new B4PrimaryGeneratorAction );  
  
    SetUserAction( new B4RunAction );  
  
    B4aEventAction* eventAction = new B4aEventAction;  
    SetUserAction( eventAction );  
  
    SetUserAction( new B4aSteppingAction( fDetConstruction,  
    eventAction ) );  
}
```

```
B4PrimaryGeneratorAction::B4PrimaryGeneratorAction() ... {
```

```
    G4int nofParticles = 1;
```

```
    fParticleGun = new G4ParticleGun( nofParticles );
```

```
    // default particle kinematic
```

```
    G4ParticleDefinition* particleDefinition
```

```
        = G4ParticleTable::GetParticleTable()->FindParticle( "e-" );
```

```
    fParticleGun->SetParticleDefinition( particleDefinition );
```

```
    fParticleGun->SetParticleMomentumDirection( G4ThreeVector(0., 0., 1. ) );
```

```
    fParticleGun->SetParticleEnergy( 50.*MeV );
```

```
}
```

```
void B4PrimaryGeneratorAction::GeneratePrimaries( G4Event* anEvent ) {
```

```
    // This function is called at the beginning of event
```

```
    ...
```

```
    // Set gun position
```

```
    fParticleGun->SetParticlePosition( G4ThreeVector( 0., 0., -worldZHalfLength ) );
```

```
    fParticleGun->GeneratePrimaryVertex( anEvent );
```

```
}
```

B4RunAction

```
B4RunAction::B4RunAction() ... {
```

```
...  
G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();  
// Book histograms, ntuple  
analysisManager->CreateH1(...);  
analysisManager->CreateNtuple(...);
```

```
...  
}
```

```
void B4RunAction::BeginOfRunAction(...) {
```

```
...  
G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();  
// Open an output file  
analysisManager->OpenFile( fileName );
```

```
}
```

```
void B4RunAction::EndOfRunAction(...) {
```

```
// Print something
```

```
...  
// Save histograms & ntuple  
analysisManager->Write();  
analysisManager->CloseFile();
```

```
}
```

```
void B4aActionInitialization::Build() const {  
    SetUserAction( new B4PrimaryGeneratorAction );  
    SetUserAction( new B4RunAction );  
    B4aEventAction* eventAction = new B4aEventAction;  
    SetUserAction( eventAction );  
    SetUserAction( new B4aSteppingAction( fDetConstruction,  
    eventAction ) );  
}
```


B4aEventAction

```
B4aEventAction::B4aEventAction( ) : ... fEnergyAbs(0.) , fEnergyGap( 0. ) ... {}
```

```
void B4aEventAction::BeginOfEventAction(...) {  
    // initialisation per event  
    fEnergyAbs = 0. ; fEnergyGap = 0.;  
    ...  
}
```

```
void B4aActionInitialization::Build() const {  
    SetUserAction( new B4PrimaryGeneratorAction );  
  
    SetUserAction( new B4RunAction );  
  
    B4aEventAction* eventAction = new B4aEventAction;  
    SetUserAction( eventAction );  
  
    SetUserAction( new B4aSteppingAction( fDetConstruction,  
    eventAction ) );  
}
```

```
void B4aEventAction::AddAbs( G4double de, G4double dl ) {  
    fEnergyAbs += de;  
    ...  
}
```

```
void B4aEventAction::AddGap( G4double de, G4double dl ) {  
    fEnergyGap += de;  
    ...  
}
```

```
void B4aEventAction::EndOfEventAction( const G4Event* event ) {  
    // Accumulate statistics: fill histograms and ntuples  
    G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();  
    analysisManager->FillH1( 1, fEnergyAbs );  
    ...  
    analysisManager->FillNtupleDColumn( 1, fEnergyAbs );  
    ...  
    // Print per-event information  
    ...  
}
```

B4aSteppingAction

```
void B4aSteppingAction::
UserSteppingAction( const G4Step* step ) {
    // Collect energy and track length step by step

    // Get volume of the current step
    G4VPhysicalVolume* volume
        = step->GetPreStepPoint()->GetTouchableHandle()->GetVolume();

    // Get energy deposit
    G4double edep = step->GetTotalEnergyDeposit();

    // Get step length
    G4double stepLength = 0.;
    if ( step->GetTrack()->GetDefinition()->GetPDGCharge() != 0. ) {
        stepLength = step->GetStepLength();
    }

    if ( volume == fDetConstruction->GetAbsorberPV() ) {
        fEventAction->AddAbs( edep, stepLength );
    }
    if ( volume == fDetConstruction->GetGapPV() ) {
        fEventAction->AddGap( edep, stepLength );
    }
}
```

```
void B4aActionInitialization::Build() const {
    SetUserAction( new B4PrimaryGeneratorAction );

    SetUserAction( new B4RunAction );

    B4aEventAction* eventAction = new B4aEventAction;
    SetUserAction( eventAction );

    SetUserAction( new B4aSteppingAction( fDetConstruction, eventAction ) );
}
```