Geant4: From Single Threading to Multi Threading

Yongchi Xiao

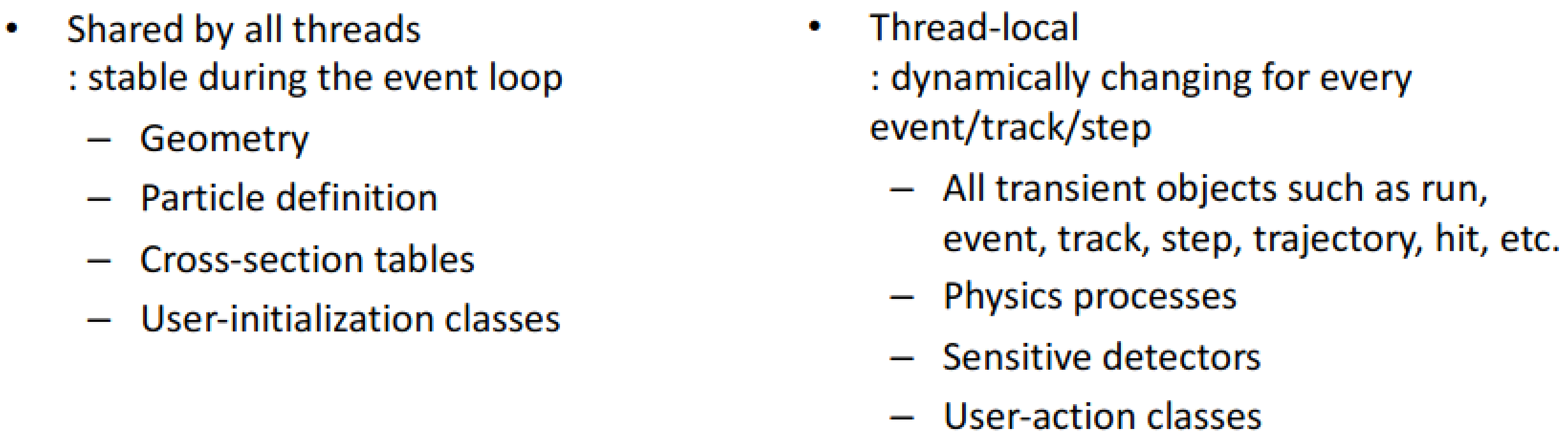
May 18th, 2021

- Useful documents:
  - Basic ideas of MT: https://indico.cern.ch/event/781244/contributions/3251900/attachments/1782717/2901032/Multithreading.pdf
  - Migration from ST to MT of existing projects: (not fully recommended) https://indico.esa.int/event/50/contributions/2583/attachments/2092/2440/HowToMigrateToMultiThread.pdf

- Customize a pre-installed MT example by Geant4
  - $G4_INSTALL/install/share/Geant4-10.6.3/examples/basic/B2/B2a

- My code with MT could be a start point:
  - /home/xiaoy/geant4/ukal2
  - Change the constructions of detectors and how to extract energy deposition in detectors as I do

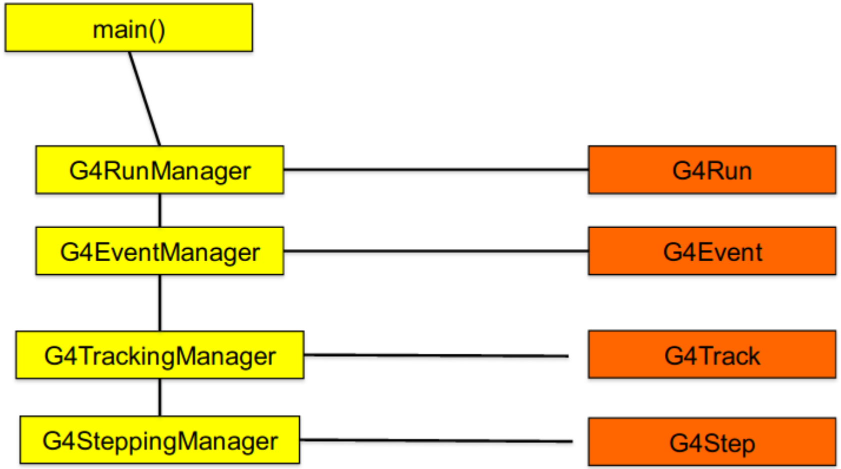
- Stable data/values during the event loop are shared (generated/stored by the master thread)
- Transient data are stored by worker threads



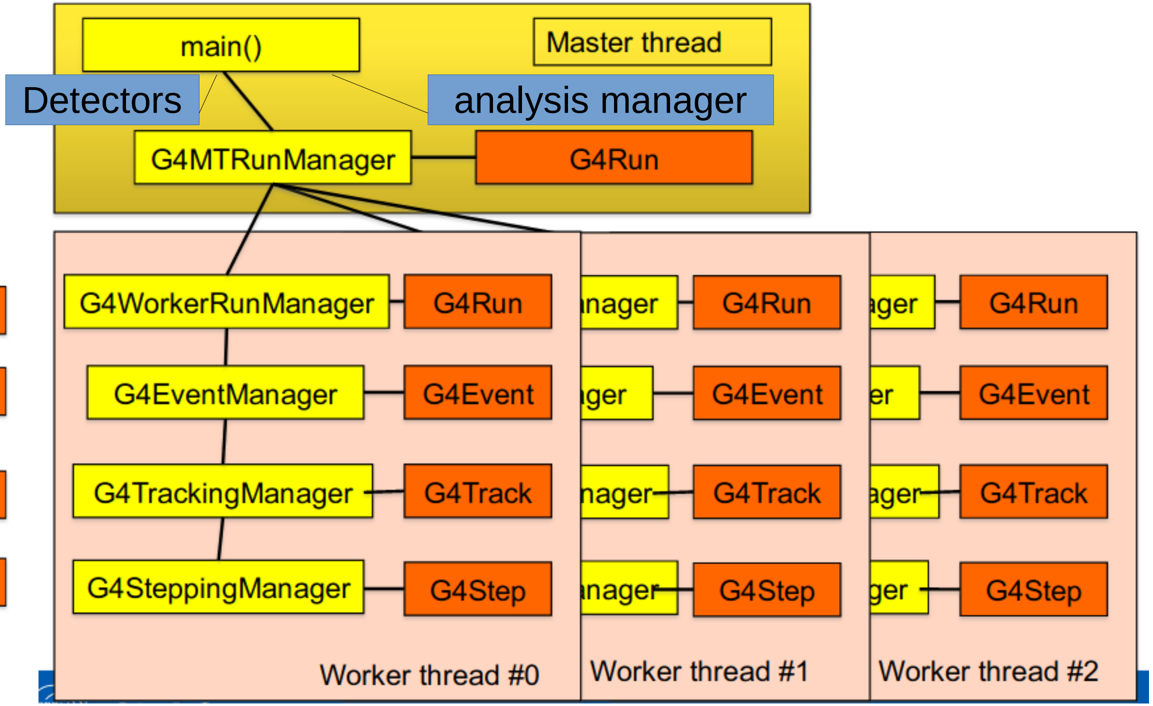
- Shared by all threads
  : stable during the event loop
  - Geometry
  - Particle definition
  - Cross-section tables
  - User-initialization classes

- Thread-local
  : dynamically changing for every event/track/step
  - All transient objects such as run, event, track, step, trajectory, hit, etc.
  - Physics processes
  - Sensitive detectors
  - User-action classes

The sequential mode:

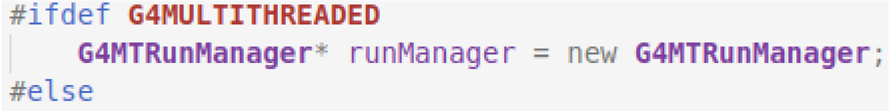


The MT mode:



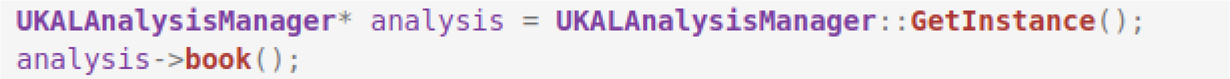
In the **main function** (defined in **exampleB2a.cc** in this case):

1. Let the code know that MT is enabled:



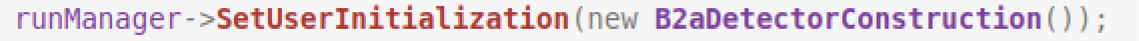
```
#ifdef G4MULTITHREADED
    G4MTRunManager* runManager = new G4MTRunManager;
#else
```

2. In order to get response of detectors (histograms), an analysis manager is needed by the master:

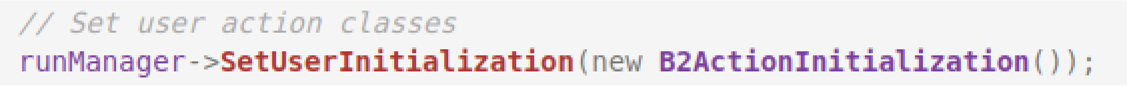


```
UKALAnalysisManager* analysis = UKALAnalysisManager::GetInstance();
analysis->book();
```

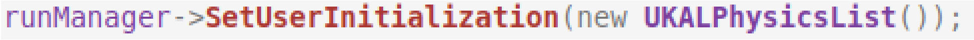
3. Do something else related to the master of MT:



```
runManager->SetUserInitialization(new B2aDetectorConstruction());
```

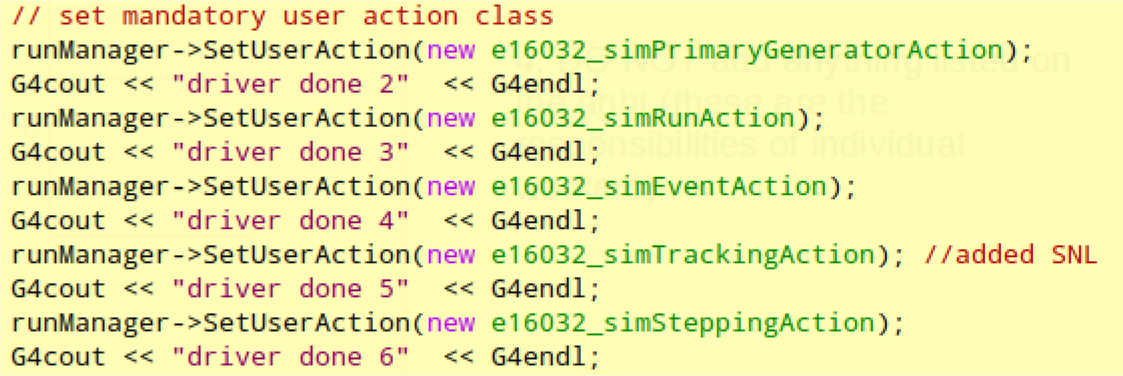


```
// Set user action classes
runManager->SetUserInitialization(new B2ActionInitialization());
```



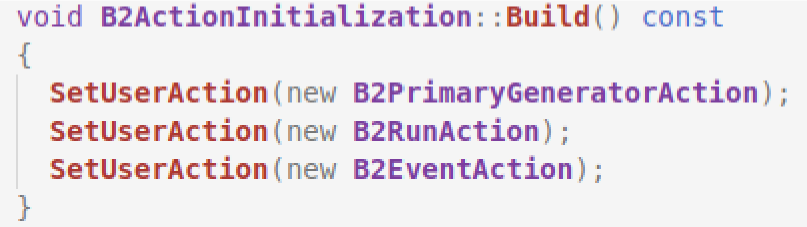
```
runManager->SetUserInitialization(new UKALPhysicsList());
```

4. DO NOT add anything listed below (these are the responsibilities of individual workers):



```
// set mandatory user action class
runManager->SetUserAction(new e16032_simPrimaryGeneratorAction);
G4cout << "driver done 2"  << G4endl;
runManager->SetUserAction(new e16032_simRunAction);
G4cout << "driver done 3"  << G4endl;
runManager->SetUserAction(new e16032_simEventAction);
G4cout << "driver done 4"  << G4endl;
runManager->SetUserAction(new e16032_simTrackingAction); //added SNL
G4cout << "driver done 5"  << G4endl;
runManager->SetUserAction(new e16032_simSteppingAction);
G4cout << "driver done 6"  << G4endl;
```

In **B2ActionInitialization.cc**:

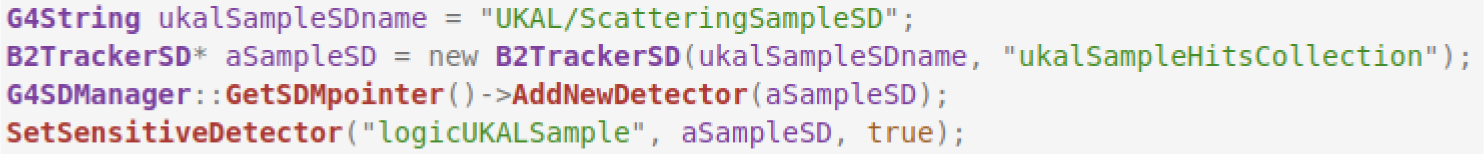


```
void B2ActionInitialization::Build() const
{
  SetUserAction(new B2PrimaryGeneratorAction);
  SetUserAction(new B2RunAction);
  SetUserAction(new B2EventAction);
}
```

In **B2aDetectorConstruction.hh/cc**:

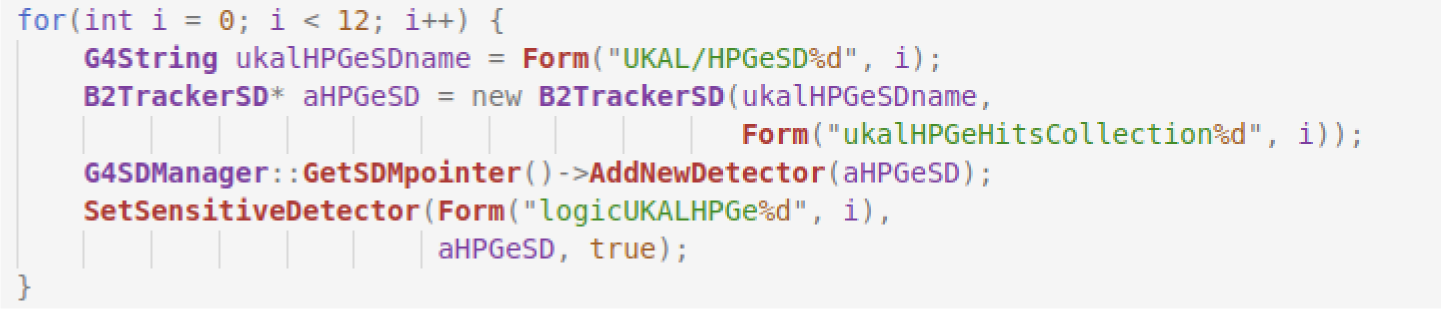
1. Define detectors as before (sensitive or not)
2. Take special care of sensitive detectors:

- There is a new function called “void ConstructSDandField()”



```
G4String ukalSampleSDname = "UKAL/ScatteringSampleSD";
B2TrackerSD* aSampleSD = new B2TrackerSD(ukalSampleSDname, "ukalSampleHitsCollection");
G4SDManager::GetSDMpointer()->AddNewDetector(aSampleSD);
SetSensitiveDetector("logicUKALSample", aSampleSD, true);
```

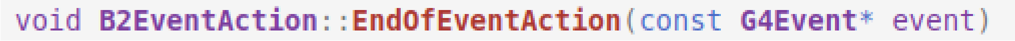
- For detectors sharing the same solid properties (HPGe etc.):
  - Assign different logic volume to each of them
  - Build physical placement for each of them
  - Construct sensitive volume for each of them:



```
for(int i = 0; i < 12; i++) {
    G4String ukalHPGeSDname = Form("UKAL/HPGeSD%d", i);
    B2TrackerSD* aHPGeSD = new B2TrackerSD(ukalHPGeSDname,
                                           Form("ukalHPGeHitsCollection%d", i));
    G4SDManager::GetSDMpointer()->AddNewDetector(aHPGeSD);
    SetSensitiveDetector(Form("logicUKALHPGe%d", i),
                         aHPGeSD, true);
}
```

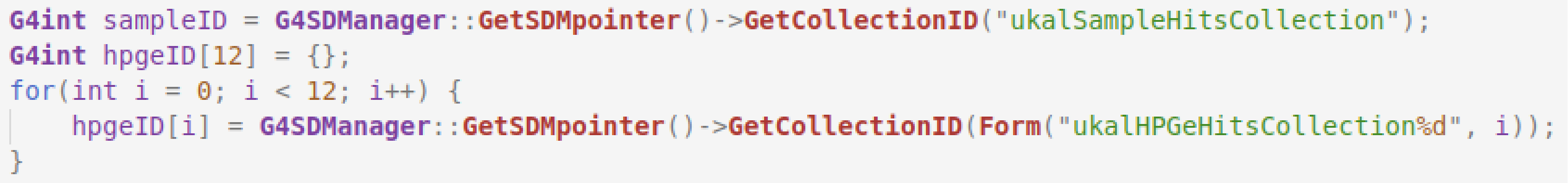
In **B2EventAction.cc**:

1. In the member function: EndOfEventAction(const G4Event* event):



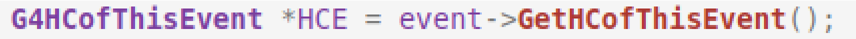
```
void B2EventAction::EndOfEventAction(const G4Event* event)
```

- Let the workers know the name of sensitive detectors and get ready to collect hits at the end of each event:

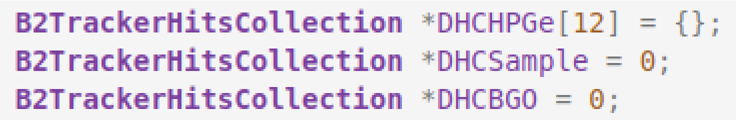


```
G4int sampleID = G4SDManager::GetSDMpointer()->GetCollectionID("ukalSampleHitsCollection");
G4int hpgeID[12] = {};
for(int i = 0; i < 12; i++) {
    hpgeID[i] = G4SDManager::GetSDMpointer()->GetCollectionID(Form("ukalHPGeHitsCollection%d", i));
}
```

- Then collect hits:

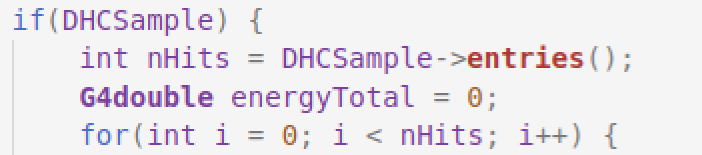


```
G4HCofThisEvent *HCE = event->GetHCofThisEvent();
```

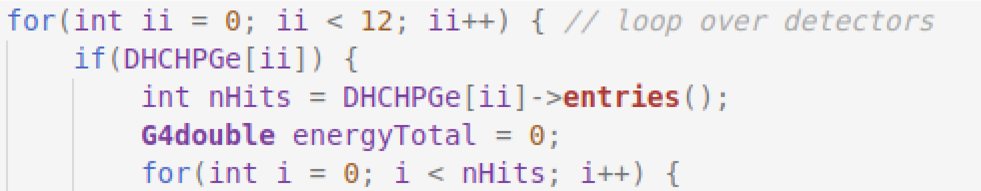


```
B2TrackerHitsCollection *DHCHPGe[12] = {};
B2TrackerHitsCollection *DHCSample = 0;
B2TrackerHitsCollection *DHCBGO = 0;
```

- Do analysis if seeing any hit in a sensitive detector:

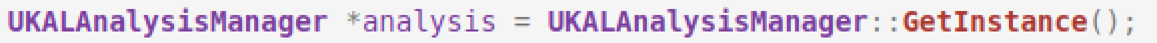


```
if(DHCSample) {
    int nHits = DHCSample->entries();
    G4double energyTotal = 0;
    for(int i = 0; i < nHits; i++) {
```

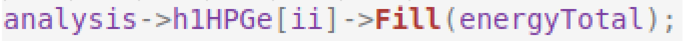


```
for(int ii = 0; ii < 12; ii++) { // loop over detectors
    if(DHCHPGe[ii]) {
        int nHits = DHCHPGe[ii]->entries();
        G4double energyTotal = 0;
        for(int i = 0; i < nHits; i++) {
```

- Call analysis manager to fill histograms at the end of each event:



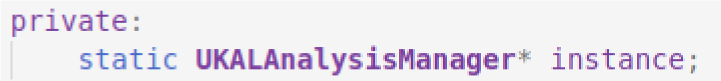
```
UKALAnalysisManager *analysis = UKALAnalysisManager::GetInstance();
```



```
analysis->h1HPGe[ii]->Fill(energyTotal);
```

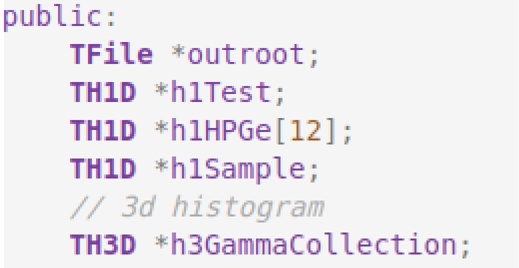
In **UKALAnalysisManager.hh** (which defines the class of the analysis manager):

A pointer to the analysis manager is needed to be called by workers



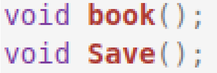
```
private:
    static UKALAnalysisManager* instance;
```

Define the histograms and make then public (accessible for workers)



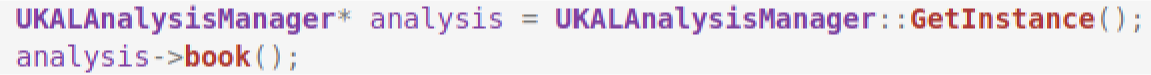
```
public:
    TFile *outroot;
    TH1D *h1Test;
    TH1D *h1HPGe[12];
    TH1D *h1Sample;
    // 3d histogram
    TH3D *h3GammaCollection;
```

Need two public functions to be called at the beginning and the end of the main function (for the master)



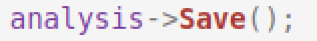
```
void book();
void Save();
```

(Recall that in the main function...)

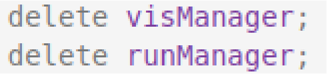


```
UKALAnalysisManager* analysis = UKALAnalysisManager::GetInstance();
analysis->book();
```

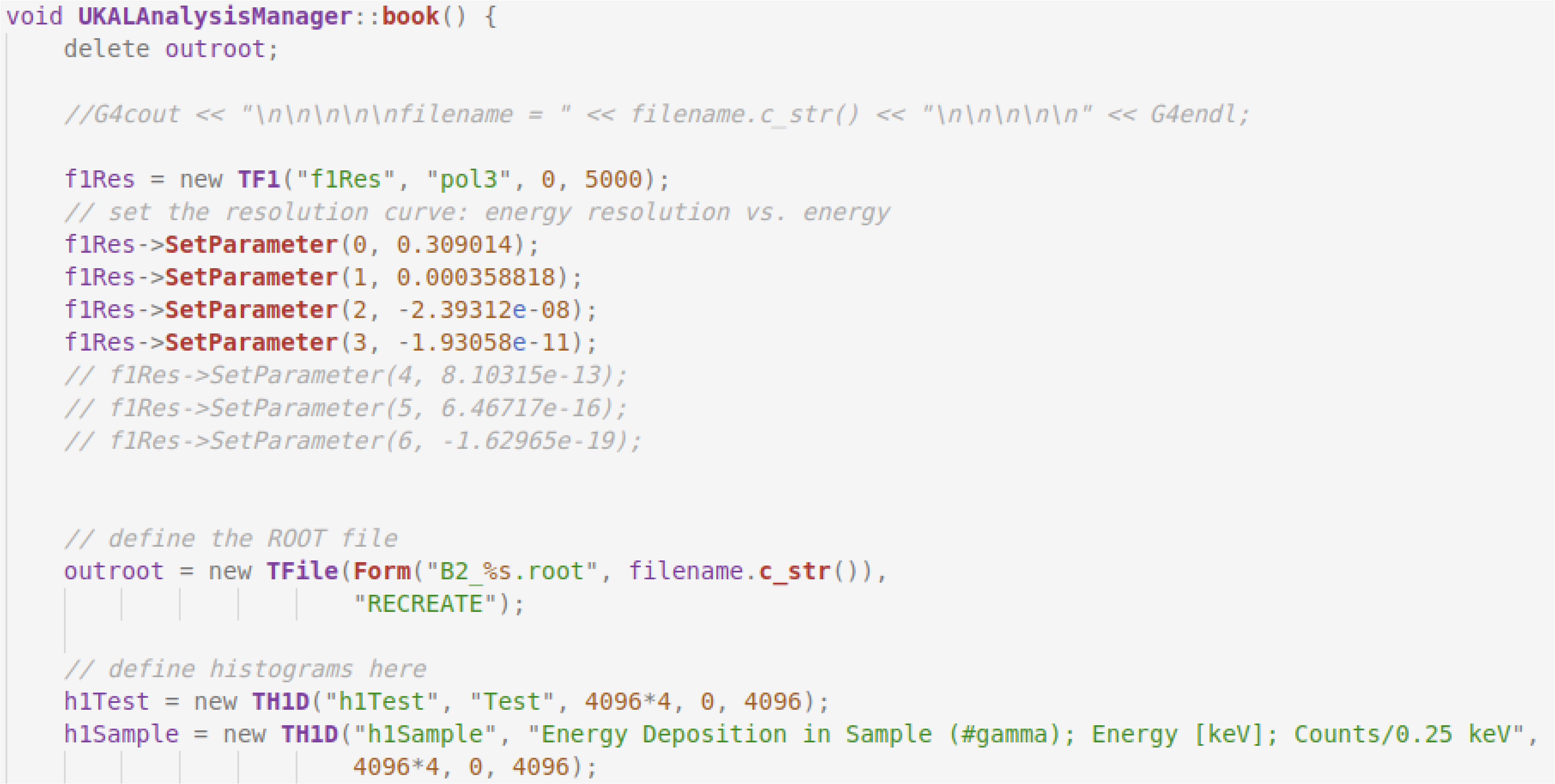
(Then before exiting the simulation...)



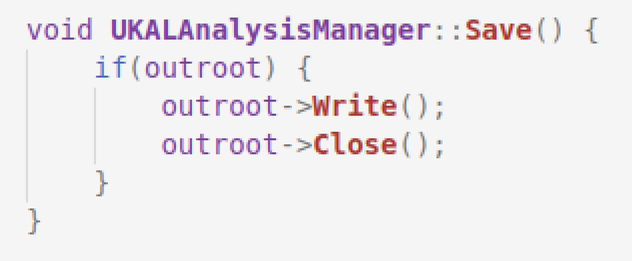
```
analysis->Save();
```



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

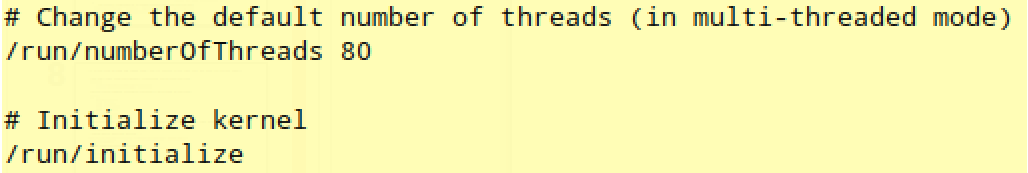


```cpp
void UKALAnalysisManager::book() {
    delete outroot;

    //G4cout << "\n\n\n\n\nfilename = " << filename.c_str() << "\n\n\n\n\n" << G4endl;

    f1Res = new TF1("f1Res", "pol3", 0, 5000);
    // set the resolution curve: energy resolution vs. energy
    f1Res->SetParameter(0, 0.309014);
    f1Res->SetParameter(1, 0.000358818);
    f1Res->SetParameter(2, -2.39312e-08);
    f1Res->SetParameter(3, -1.93058e-11);
    // f1Res->SetParameter(4, 8.10315e-13);
    // f1Res->SetParameter(5, 6.46717e-16);
    // f1Res->SetParameter(6, -1.62965e-19);


    // define the ROOT file
    outroot = new TFile(Form("B2_%s.root", filename.c_str()),
                        "RECREATE");

    // define histograms here
    h1Test = new TH1D("h1Test", "Test", 4096*4, 0, 4096);
    h1Sample = new TH1D("h1Sample", "Energy Deposition in Sample (#gamma); Energy [keV]; Counts/0.25 keV",
                        4096*4, 0, 4096);
```



```cpp
void UKALAnalysisManager::Save() {
    if(outroot) {
        outroot->Write();
        outroot->Close();
    }
}
```

In the command macro:



```
# Change the default number of threads (in multi-threaded mode)
/run/numberOfThreads 80

# Initialize kernel
/run/initialize
```