# Tabular Editors for Geant4: Geant4 Geometry Editor and Geant4 Physics Editor

*H. Yoshida*[1], *T. Kodama*[1], *M. Nagamatu*[1], *S. Sei*[1], *T. Yamada*[1], *H. Kurasige*[2]

[1] Laboratory of Technology Education, Naruto University of Education, Japan
[2] Department of Physics, Kobe University, Japan

**Abstract**

We have developed two tabular editors for Geant4, i.e., Geant4 Geometry Editor and Physics Editor. They provide for a user, his own implementation of user mandatory C++ classes which must be derived from the two base classes of Geant4 for geometry and physics. They are implemented with Java-2 and run on Linux and Windows.

Keywords: Geant4, tabular editor, geometry, physics

## 1 Why Tabular Editors?

To use Geant4, user must define three mandatory classes, two of which are user initialization classes, and the other is the user action class. The base classes of these mandatory classes are abstract, and Geant4 does not provide default behavior for them. User must inherit from the abstract base classes and implement his own classes.

One initialization class is **G4VUserDetectorConstruction** in which the complete detector setup should be described. Another is **G4VUserPhysicsList** in which user must specify all particles and physics processes used in his simulation.

GGE (Geant4 Geometry Editor) and GPE (Geant4 Physics Editor) have for their purpose to help a user to implement his concrete classes. The editors have a look of spreadsheets with fixed numbers of columns and with as many rows as filled by a user. From the contents of the table, the concrete class files can be automatically generated. The tables can be saved in the Java's persistent files and can be loaded for reuse. The editors are intended to make him free of complex usage of classes and of straight-forward but tedious typing of names of many classes and methods.

In designing the editors, we have to decide the trade-offs between their feasibility and wider range of their applicability. While the general patterns of implementing the derived classes can be found in some parts, some typical application patterns must be assumed to give shape to the design.

## 2 GGE: GEANT4 Geometry Editor

GGE consists of the material editor and the volume editors for logical and physical volumes. While GGE provides the capability of defining compound materials and logical volumes in a general way, its capability of defining physical volumes is limited to relatively simple spatial arrangements as described later.

### 2.1 Material Editor

Materials (chemical compounds, mixtures) made of elements are supported by GGE. At present, it doesn't support elements made of isotopes. Material editor is composed of two independent tables;

the table for materials from scratch with the columns (Use, Name, A, Z, density and unit, state, temperature and unit, pressure and unit) and one for compound materials with the columns (Use, Name, Elements, density and unit, state, temperature and unit, pressure and unit). By clicking the Elements column, a new window is open to input a composition of the compound material with either fractions by weight or numbers of atoms. The periodic table of elements is popped-up to select one (for scratch) or multiple elements (for compound) by mouse click(s).

From each row marked as *Use*, C++ code for instances of **G4Element** and **G4Material** classes are generated. The material table can be saved in a persistent file.

### 2.2 Volume Editors ; logical and physical volumes

A detector geometry is made of a number of volumes. The largest volume, i.e., the world volume contain all other volumes. Daughter volumes can be placed within another mother volume. With GGE, such hierarchy of volumes can be defined to create complex geometry.

#### 2.2.1 Logical volume editor

Each row of the table of the logical volume with four columns (volume's name, solid, material, VisAttribute) corresponds to the mandatory arguments to make an instance of the **G4LogicalVolume** class. Other arguments; G4FieldManager, G4VSensitiveDetector and G4UserLimits are set to the defaults.

Nine CSG and two BREP solids can be selected from the list-box. The guided type-in cells for the respective solids facilitates user's input of their parameters and canonical units. A solid can be previewed by **DAWN** renderer. GGE doesn't support Boolean operation between solids.

From each row in the logical volume table are generated C++ codes for its solid, logical volume and its visualization attribute.

#### 2.2.2 Physical volume editor

From the **G4PVPlacement** class, GGE can derive a physical volume of single positioning and of repeated positioning of a volume with the identifying copy numbers. GGE provides two types of spatial arrangement of volumes; translation along an axis and axial rotation. Mother volume can be either logical or physical. Frame or body rotation can be selected. GGE implements **G4PVReplica** class to have multiple identical copies along the Cartesian or cylindrical coordinates.

We take the case of "Repeated positioning of a volume" for the illustration.

A translational arrangement is defined by a row with thirteen columns; the name of the physical and logical volume, the type of the mother volume, the name of the mother volume, the position of the first copy, the direction of translation (X, Y or Z axis) and incremental step size.

An axially rotational arrangement is defined by a row with sixteen columns; the type of rotation, the name of the physical and the logical volume, the type and the name of the mother volume, position of the center of axial rotation, rotational axis (parallel to X, Y or Z), the radius of rotation, the starting angle and incremental step angle, and the number of copies.

From each row in the table are generated a chunk of C++ codes to initialize the "for" loop with the number of copies, to specify the amount of translation and rotation for the i-th copy, and to make instances of **G4PVPlacement** located at the i-th **G4Transform3D**-ed position.

### 2.3 Generation of C++ code

The pure virtual method **Construct()** of the base class **G4VUserDetectorConstruction** must be implemented in the user's concrete class for his geometry. GGE generates, in sequence, groups of C++ codes for G4Elements, G4Materials, G4VisAttributes, G4Solids, G4LogicalVolumes, Single

Positioned Volumes, Repeated Volumes, and Replicas. The order of instantiation inside each group is decided by the tables filled by the user. Finally the name of the world volume must be returned by **Construct()**. GGE output the C++ code to a editor widget in which all necessary header files are automatically included.

### 2.4 Compile and visualize

GGE is supplemented with the minimum set of class files to make a Geant4 executable for visualization. To have the whole view of the detector, user have to follow à-la-TEX scheme, that is, to compile the generated C++ codes, to link with the Geant4's libraries and to use its visualization system. An integrated GUI called Momo offers menu buttons to help a user with a rapid iteration to use GGE, to compile and to use GAG (Geant4 Adaptive GUI) to control visualization.

## 3 GPE: Geant4 Physics Editor

GPE helps a user to implement three pure virtual methods derived from the mandatory base class **G4VUserPhysicsList**: **ConstructParticle()** to construct particles, **ConstructPhysics()** to construct processes and register them to particles and **SetCuts()** to set a cut value in range to all particles.

GPE consists of two physics tables, one for electromagnetic processes and other for hadronic processes. Each physics table has two buttons to pop up the particle table and the process table proper to it. Among seven major process categories: "electromagnetic", "hadronic", "transportation", "decay", "optical", "photolepton-hadron", and "parameterization", the category of electromagnetic processes is implemented in GPE. We have decided that GPE incorporates, by default, the transportation and decay processes. Hadronic processes with various models is under construction.

Parameterization and optical processes are not implemented due to their close connection to the detector geometry.

User has to define the name of his concrete class and to specify the default cut value. The table as well as all user-defined objects are saved in a persistent file and can be loaded for reuse.

### 3.1 Particle Tables

GPE has two particle tables, one for electromagnetic processes and another for hadronic processes. The former has buttons for all particles available in Geant4, while the latter has much less buttons only for long-lived mesons, baryons and ions.

To implement **ConstructParticle()**, user can choose a particle or the whole set of particles belonging to a category from the particle table. Selected particles are automatically copied to the physics table. Particles in the category "Shortlived" can be either chosen all at once or not chosen at all.

### 3.2 Process Tables

Each process must have information on applicable particle types and default ordering parameters for each **DoIt** method. Each hadronic process must have model(s) registered to it. A model must have information on applicable particle types and processes.

A row of the electromagnetic physics table consists of five columns; particle, process, ordering parameters (AtRest, AlongStep and PostStep). The table of the electromagnetic processes contains nineteen classes of "standard" electromagnetic processes, six classes for muons, two classes related with X rays and seven low energy processes. Upon selection of a process from

the table of electromagnetic processes, the process is copied to the physics table with the default ordering parameters.

The button for the default transportation and decay processes is provided, too.

The table of hadronic processes has twenty nine buttons for inelastic processes of long-lived hadrons and ions, hadron capture, hadron fission and seven buttons for stopping processes.

### 3.3 Generation of C++ code

**ConstructParticle()** methods consists of **ConstructBosons()**, **ConstructLeptons()**, **ConstructMesons()**, **ConstructBaryons()**, **ConstructIons()** and **ConstructShortLiveds()**, corresponding to six categories of particles in Geant4. Scanning the physics table, GPE includes all necessary header files of particles. Generation of C++ codes for **ConstructParticle()** is straightforward.

Scanning the physics table, GPE includes required process headers to **ConstructPhysics()**. **ConstructPhysics()** method consists of **ConstructEM()** for electromagnetic processes, **ConstructGeneral()** for decays, **ConstructHad()** for hadronic processes etc.. For **ConstructEM()**, GPE generates codes of the loop for the particle iterator. Then, it generates codes within the loop to register a process to the process manager by specifying the pointer to the process object and ordering parameter for the process, for example;

```
if (particleName == ``e-'') {
  pmanager->AddProcess(new G4MultipleScattering(),ordInActive,1,1);
}
```

The C++ codes of **ConstructGeneral()** for decay processes and the transportation are hard coded by GPE. The default cut value user have typed in is used to implement **SetCutsWithDefault()**.

## 4 Implementation of GGE and GPE

We have chosen Java because of its Swing set, serialization and platform-independence. As their specifications have been evolving rapidly, GGE and GPE have had to pass through many revisions. At present they run with JDK-1.2.2 on Linux and Windows.

## References


1 "Geant4 User's Document" http://wwwinfo.cern.ch/asd/geant4/G4UsersDocuments/Overview/html/index.html

2 M. Nagamatsu et al., "GAG: GEANT4 Adaptive Graphical User Interface", CHEP'98, Chicago, Autumn 1998.

3 "Geant4 GUI at Naruto" http://erpc1.naruto-u.ac.jp/~geant4/