

MANCHESTER  
1824

The University of Manchester



*Geant4* ASSOCIATES  
INTERNATIONAL  
*Experts in Radiation Simulation*

# The Geant4 User Interface and Visualisation Part 1

Introduction  
Installing and building  
A basic app  
The Qt GUI

# INTRODUCTION

Contributors

Aims

Available User Interfaces and  
Graphics Drivers

# Contributors

- Main contributors listed (with approximate start year) – but there is much cross-fertilization, including from the Geometry Category
- **UI systems**
  - Core framework: Makoto Asa (1994), Hajime Yoshida, Koichi Murakami, and Guy Barrand (1998)
  - Xm GUI: Guy Barrand (1998)
  - Qt GUI: Laurent Garnier (2007), Guy Barrand (2007) and John Allison (2023)
  - Win32 GUI: Guy Barrand (1998), O. Pena-Rodrigue and Gabriele Cosmo (2021)
- **Graphics Reps** (low-level library of graphics primitives, etc)
  - Core primitives and vis manager virtual interface: John Allison (1994)
  - Polyhedron and Boolean processors: Evgueni Tchernae (1998)
- **Modeling**
  - Core models: John Allison (1994)
  - Trajectory models and filters: Jane Tinslay (2005)
- **Graphics**
  - Core vis manager: John Allison (1994)
  - OpenGL drivers
    - Core scene handling: Andy Walkden and John Allison (1996)
    - Qt extension: Laurent Garnier (2007)
  - Open Inventor drivers: Joe Boudreau (1996), Guy Barrand (2004), Fred Jones (2012)
  - Ray Tracer: Makoto Asai (2000)
  - DAWN and VRML: Satoshi Tanaka (1996)
  - HepRepFile: Joseph Perl (2001)
  - VTK drivers: Stewart Boogert (2021)
  - ToolsSG drivers: Guy Barrand (2021)
  - ASCII Tree: John Allison (2001)
  - Others...

# Aims

- The Geant4 UI and vis systems have evolved over many years
- The primary aim: to help G4 app developers
  - Check the geometry
  - Check the interactions
  - Understand and validate their app
  - Provide a custom app for your users
- Secondly, to take advantage of modern graphics libraries
  - An object-oriented interface
  - A multi-functional manager designed around the users' needs
- Each UI and graphics system, and even the manager itself, are effectively plug-ins, and an ambitious user may write his/her own to the abstract interfaces

# Available User Interfaces and Graphics Drivers

- **User Interfaces**

- **Plain terminal**

- csh & tcsh

- **Graphical**

- Xm (motif)
  - In-window graphics, interactive help, and output
- Win32 (Windows)
  - As above
- Qt (cross-platform UI system)
  - As above, **plus** an interactive scene tree
    - Select visibility and colour of geometry objects...and more (described later)

- **Graphics Drivers**

- **Graphical**

- OpenGL (some advanced features, also export)
- OpenInventor (some advanced features, VRLM export)
- ToolsSG (GLES/Qt/ZB, export and plotting)
- Qt3D (prototype)
- VTK (many interactive options and export formats)
- RayTracerX (export jpeg)
- ...

- **File-writing** (always registered):

- DAWNFILE (browse with DAWN)
- HepRepFile (browse with HepRApp)
- VRML2FILE (browse with any web browser)
- RayTracer (export jpeg)
- ToolsSGOffscreen (numerous picture formats)

- **Diagnostic** (always registered):

- ASCIITree (always registered, dumps to G4cout)

# BASICS

- Building your app, installing libraries, Geant4 build options
- Instantiating the UI and vis manager
- Choosing the UI and graphics driver
- Vis concepts
- Class diagram
- Vis attributes
- View parameters

# Installing Qt

- MacOS: `brew install qt`
  - For Qt5: `brew unlink qt` (to avoid linking to Qt6)
- Linux: `apt install qtbase5-dev`
- Windows:
  - Binaries from Qt web site (qt.io)
  - Does `conda` work?

## Building with Qt

- You will get the Qt GUI
- You get several graphics drivers
  - OGLI, OGLS, TSG-GLES, Qt3D
- (You can choose at run-time – see later)
- Options in red for Qt6 and Geant4 11.3
- “-G Xcode” only for Mac Xcode users

```
cmake -Wno-dev --log-level=ERROR \  
-DCMAKE_INSTALL_PREFIX=`pwd` \  
-DGEANT4_USE_GDML=ON \  
-DGEANT4_USE_QT=ON \  
-DGEANT4_USE_QT_QT6=ON \  
-DCMAKE_PREFIX_PATH=\  
"$(brew --prefix qt@6)" \  
-DGEANT4_ENABLE_TESTING=ON \  
-DGEANT4_USE_FREETYPE=ON \  
-G Xcode \  
~/Geant4/geant4-dev
```

# Other libraries

- Installing X11
  - MacOS: Xquartz
  - Linux: native
  - Windows: X11 not available, but special drivers use the native windowing system.
- Installing VTK
  - MacOS: **brew install vtk**
  - Linux: **apt install vtk**
  - Windows: **conda install vtk**
- Installing Coin3D (Open Inventor)
  - Coin3D (Open Inventor): See <https://www.coin3d.org>



# The UI and vis build options

- From the [Installation Guide](#), [Build Options](#):
  - GEANT4\_USE\_INVENTOR (DEFAULT : OFF)
  - GEANT4\_USE\_INVENTOR\_QT (DEFAULT : OFF)
  - GEANT4\_USE\_OPENGL\_WIN32 (DEFAULT : OFF, Windows Only)
  - GEANT4\_USE\_OPENGL\_X11 (DEFAULT : OFF, Unix Only)
  - GEANT4\_USE\_QT (DEFAULT : OFF)
  - **GEANT4\_USE\_QT\_QT6 (DEFAULT : OFF)**
  - GEANT4\_USE\_RAYTRACER\_X11 (DEFAULT : OFF, Unix only)
  - GEANT4\_USE\_VTK (DEFAULT : OFF)
  - GEANT4\_USE\_XM (DEFAULT : OFF, Motif, Unix Only)
- **Don't mix X11 and Qt** (unless you know what you're doing!!)
- GEANT4\_USE\_QT brings in:
  - OpenGLQt
  - ToolsSGQt
  - Qt3D (under development)

# main()

- A minimal interactive app
- A simple vis.mac
- And gps.mac
- Uses G4GeneralParticleSource

```
int main(int argc, char** argv) {  
    auto ui = new G4UIExecutive(argc, argv);  
    auto runMan = G4RunManagerFactory::CreateRunManager();  
    runMan->SetUserInitialization(new DetectorConstruction);  
    auto physList = G4PhysListFactory().ReferencePhysList();  
    runMan->SetUserInitialization(physList);  
    runMan->SetUserInitialization(new ActionInitialization);  
    auto visMan = new G4VisExecutive;  
    visMan->Initialise();  
    auto Ulmanager = G4Ulmanager::GetUlpointer();  
    Ulmanager->ApplyCommand("/control/execute vis.mac");  
    ui->SessionStart();  
    delete ui;  
    delete visMan;  
    delete runMan;  
}
```

```
gps.mac  
-----  
/gps/particle proton  
/gps/position 0 0 -20 cm  
/gps/direction 0 0 1  
/gps/energy 70 MeV
```

```
vis.mac  
-----  
/control/verbose 2  
/run/initialize  
/vis/open  
/vis/drawVolume  
/vis/viewer/set/viewpointThetaPhi 110 170 deg  
/vis/scene/add/axes  
/vis/scene/add/trajectories  
/vis/scene/endOfEventAction accumulate  
/control/execute gps.mac
```

# Printed at start-up

- On instantiation of **G4UIExecutive**
  - Before instantiation of actual session, so on `std::cout`
- On instantiation of **G4VisExecutive**
  - **Some drivers always registered**
    - They do not need an external graphics library

Available UI session types: [ Qt, tcsh, csh ]

You have successfully registered the following graphics systems.  
Registered graphics systems are:

ASCIITree (ATree)  
DAWNFILE (DAWNFILE)  
G4HepRepFile (HepRepFile)  
RayTracer (RayTracer)  
VRML2FILE (VRML2FILE)  
gMocrenFile (gMocrenFile)  
TOOLSSG\_OFFSCREEN (TSG\_OFFSCREEN, TSG\_FILE)  
OpenGLImmediateQt (OGLIQt, OGLI)  
OpenGLStoredQt (OGLSQt, OGL, OGLS)  
Qt3D (Qt3D)  
TOOLSSG\_QT\_GLES (TSG\_QT\_GLES, TSGQt, TSG)  
TOOLSSG\_QT\_ZB (TSG\_QT\_ZB, TSGQtZB)

You may choose a graphics system (driver) with a parameter of the command `/vis/open` or `/vis/sceneHandler/create`, or you may omit the driver parameter and choose at run time:

- by argument in the construction of `G4VisExecutive`;
- by environment variable `"G4VIS_DEFAULT_DRIVER"`;
- by entry in `"~/g4session"`;
- by build flags.

- Note: This feature is not allowed in batch mode.

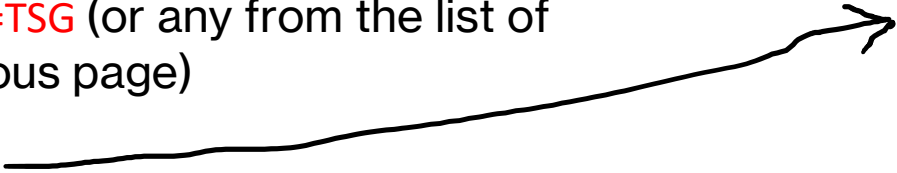
For further information see `"examples/basic/B1/exampleB1.cc"` and `"vis.mac"`.

Always registered

Depending on options

# Choosing the UI and Vis Driver at run time

- If you do nothing before start-up, the “best” UI and Vis are chosen according to your original build flags
- Choosing with environment variables
  - UI: `G4UI_USE_TCSH=1`, `G4UI_USE_WIN32=1`, `G4UI_USE_XM=1`, or `G4UI_USE_QT=1`
  - Vis: `G4VIS_DEFAULT_DRIVER=TSG` (or any from the list of registered drivers – previous page)
- Using `~/.g4session`
  - This is a dot (.) file you keep in your home directory (~/)
  - The format is:
    - First line: chosen default UI session: `Qt`, `Xm`, `Win32`, `tcsch` or `csch`
    - Subsequent lines: `<app> <session> [<vis>] [<window-size>]`
- Note: Environment takes precedence.
- After all the above you can still change driver with `/vis/open [driver]`



```
Qt # Default session
#exampleB1 tcsch
#exampleB1 Qt TSG 1000x1000+0-0
#exampleB1 Qt Qt3D
#exampleB2a tcsch
exampleB2b Qt Qt3D
#exampleB2b tcsch
#exampleB3 tcsch
#exampleB4a Xm
exampleB5 Qt TSG
#test202 tcsch
#test202 Qt VTK
#userVisAction tcsch
```

# The Qt GUI

- **Help**

- Shows selection of `/vis/scene/list`
  - Double-click to transfer to Session window
  - Then complete as required and hit return

- Note: commands are highlighted (if echoed with `/control/verbose 2`)

- Several single action buttons/icons along the top

The screenshot shows the Geant4 Qt GUI interface. The main window displays a 3D scene with a blue background. A cylinder labeled 'Shape1' and a rectangular block labeled 'Shape2' are visible. A green 'G4' logo is also present. The scene is titled 'exampleB1' and the date 'Sun Jul 21 12:12:33 2024' is shown in the top right corner. The 'Geant4' logo is in the bottom left, and 'exampleB1' is in the bottom right.

The left sidebar contains a 'Scene tree' with a search bar and a list of commands. The 'list' command is highlighted. Below the list, the command `/vis/scene/list` is shown with its guidance: 'Lists scene(s). "help /vis/verbose" for definition of verbosity.' A table below the command shows parameters:

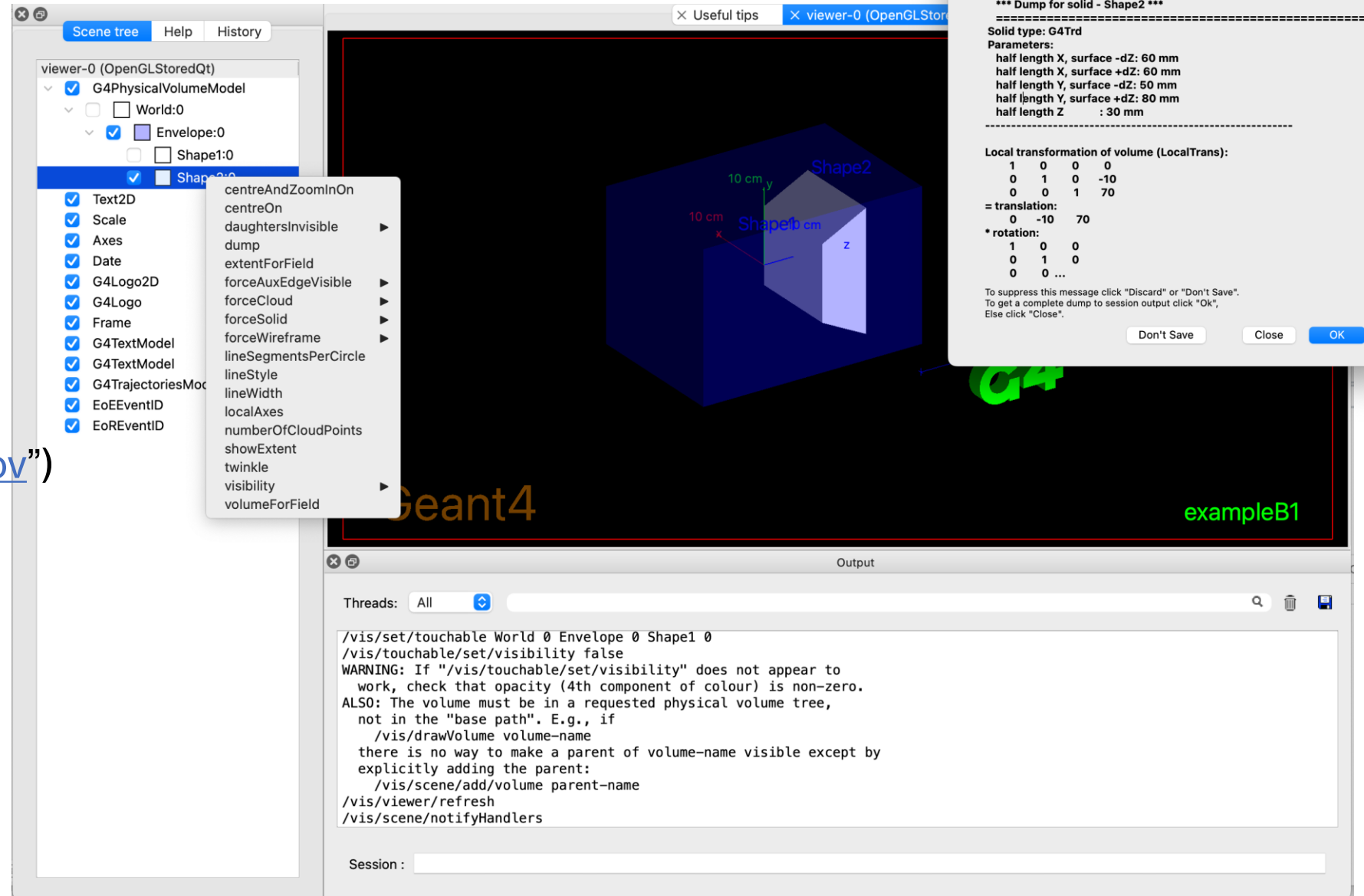
	Parameter	Guidance	Type	Omittable	
1	scene-name		s	True	al
2	verbosity		s	True	w

The bottom window is the 'Output' terminal, showing the output of the `/vis/scene/list` command:

```
/vis/scene/list
(current) scene "scene-0"
Run-duration models:
Active: G4PhysicalVolumeModel World:0 BasePath: TOP
Active: Text2D: 'exampleB1' at 0.9,-0.9 with size 24 with offsets 0,0
Active: Scale: 1 auto auto 1 0 0 auto 0 0 0 m
Active: Axes: 0 0 0 -1 m auto true
Active: Date: 18 0.95 0.9 right -
Active: G4Logo2D: 48 -0.9 -0.9 left
Active: G4Logo: 1 auto auto 0 1 0 auto 0 0 0 m
Active: Frame: 0.97
Active: G4TextModel: 'Shape1' at 0 6 -4 cm with size 18 with offsets 4,4
Active: G4TextModel: 'Shape2' at 6 7 10 cm with size 18 with offsets 4,4
End-of-event models:
Active: G4TrajectoriesModel for any type of trajectory
Active: EoEventID: 18 -0.95 0.9 left
End-of-run models:
Active: EoEventID: 18 -0.95 0.9 left
```

# The Qt GUI (contd)

- Scene tree
  - Click to make (in)visible, or change colour
  - Right-click to get other actions (see video [“Scene tree centreAndZoomInOn.mov”](#))
- Plus, of course, you get an interactive graphical window
- [Example B1 demo](#)



# Adding your own action buttons or icons

- The Qt GUI has several built-in buttons



- You can remove, then add back what you want

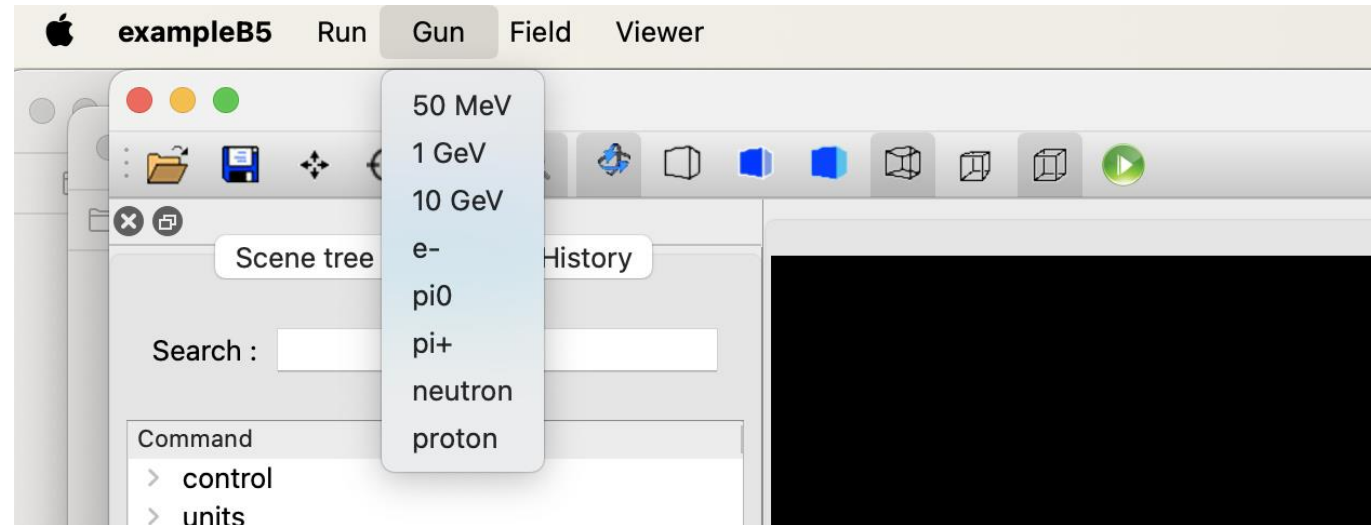
- `/gui/defaultIcons false`
- `/gui/addIcon "Move" move`
- See `examples/basic/B5/icons.mac`

```
UImanager->ApplyCommand("/control/execute init_vis.mac");  
if (ui->IsGUI()) {  
    UImanager->ApplyCommand("/control/execute gui.mac");  
}
```

- You can add your own `gui.mac` (applies also to the the Xm GUI)

- See basic examples B2, B4 and B5

```
/gui/addMenu gun Gun  
/gui/addButton gun "50 MeV"    "/gun/energy 50 MeV"  
/gui/addButton gun "1 GeV"     "/gun/energy 1 GeV"  
/gui/addButton gun "10 GeV"    "/gun/energy 10 GeV"
```



# Back-up slides



# A PrimaryGeneratorAction

- Uses **G4GeneralParticleSource**
- Instantiated in **PrimaryGeneratorAction**
  - Itself instantiated in **ActionInitialization**

```
#ifndef PrimaryGeneratorAction_hh
#define PrimaryGeneratorAction_hh 1
#include "G4VUserPrimaryGeneratorAction.hh"
class G4GeneralParticleSource;
class PrimaryGeneratorAction :
public G4VUserPrimaryGeneratorAction {
public:
    PrimaryGeneratorAction();
    ~PrimaryGeneratorAction();
    void GeneratePrimaries(G4Event*);
private:
    G4GeneralParticleSource* fpParticleGun;
};
#endif
```

```
#include "PrimaryGeneratorAction.hh"
#include "G4GeneralParticleSource.hh"
PrimaryGeneratorAction::PrimaryGeneratorAction()
{fpParticleGun = new G4GeneralParticleSource;}
PrimaryGeneratorAction::~~PrimaryGeneratorAction()
{delete fpParticleGun;}
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{fpParticleGun->GeneratePrimaryVertex(anEvent);}
```