g2MIGTRACE Tutorial: Storing Output

Kevin Lynch
June 2011
Outline

• What already gets output
• Users:
  • Getting g2MIGTRACE to output something
  • Doing something useful with it all
• Developers:
  • How it all works
  • How to add output
So, what does get output?

- Short answer:
  - Nothing!
- Longer answer:
  - If you exert de minimis effort, there are a few things you can get...
Longest answer

- Simulation Metadata
- Run Level Data
  - Collected Event Level Data
    - Particle Level Data
Simulation Metadata

- SVN Revision
- Git revision
- Build time
- External library revisions
Run Level Data

- Physical Object Manager
- *Configuration Parameters*
Event Level Data

- Particle ID Data
- Final Event Status Data
- Inflector Tracking Data
- Ring Tracking Data
  - Spin Tracking Data
- Energy Loss Data
- Calorimeter Hit Data
- Hodoscope Hit Data
- Wire Chamber Hit Data
What does this look like

[krlynch@i-m-so-tired Linux-g++]$ ls
g2MacroFiles/  g2MIGTRACE_20100710015359_run0000.root  g2StudyMacros/
g2MIGTRACE*  g2RunTimeFiles/
[krlynch@i-m-so-tired Linux-g++]$
Inside the Root file

```
root [3] .ls
TFile**    ./g2MIGTRACE_20100710015359_run0000.root   g2MIGTRACE output file
TFile*     ./g2MIGTRACE_20100710015359_run0000.root   g2MIGTRACE output file
KEY: TParameter<int> svn_revision;1 Named templated parameter type
KEY: g2UniqueObjectManager  uom;1
KEY: TTree trackerTree;1 Ring Beam Tracking Data
KEY: TTree trackTree;1    Track ID Birth Data
KEY: TTree eventStatusTree;1 Event final status data
```
Users

- Getting g2MIGTRACE output
- Doing something useful with it
  - Aigh! There's the rub....
Getting some output

g2MIGTRACE >> ls /g2MIGTRACE/rootStorage/
Command directory path : /g2MIGTRACE/rootStorage/

Guidance :
Root File Storage Management

Sub-directories :
/g2MIGTRACE/rootStorage/g2UOM/ Unique Object Manager Query Commands
/g2MIGTRACE/rootStorage/ringHits/ Ring hit monitor controls

Commands :
storageStatus * Determine whether a Root file is created to store simulation results
basename * Get/set the base filename: <basename>_<datetime>_run#.root; omit arg for "get"
outdir * Get/set the output directory; omit arg for "get"
inflctorTrackerStatus * Get/set the inflector tracker enable state; omit arg for "get"
ringTrackerStatus * Get/set the ring tracker enable state; omit arg for "get"
ringHitStatus * Get/set the ring hit monitor enable state; omit arg for "get"
caloHitStatus * Get/set the calorimeter hit monitor enable state; omit arg for "get"
g2MIGTRACE >>
Let's enable Root output

1. g2MIGTRACE >> /g2MIGTRACE/rootStorage/storageStatus
   /g2MIGTRACE/rootStorage/storageStatus
   A Root file will not be output

2. g2MIGTRACE >> help /g2MIGTRACE/rootStorage/storageStatus
   Command /g2MIGTRACE/rootStorage/storageStatus
   Guidance:
   Determine whether a Root file is created to store simulation results
   Parameter: Choice
   Parameter type: s
   Omittable: True
   Candidates: on off get

3. g2MIGTRACE >> /g2MIGTRACE/rootStorage/storageStatus on
   /g2MIGTRACE/rootStorage/storageStatus on

4. g2MIGTRACE >> /g2MIGTRACE/rootStorage/storageStatus
   /g2MIGTRACE/rootStorage/storageStatus
   A Root file will be output
   g2MIGTRACE >>
What do you actually get now?

- A Root File
  - with a long name!
- g2MIGTRACE metadata
  - svn_revision
- The object manager
  - G2UniqueObjectManager ... on which more later
- Particle Data Holder
  - TrackTree
- Event Status Data Holder
  - EventStatusTree
- More on all of these later...
If you want more, you must

- Enable it!
- Write it ... then enable it!
What can you enable?

- Beam Tracking output
  - Ring tracking
  - Inflector tracking
- “Energy Loss” output
  - Generic energy loss
    - Ring hits
  - Specialized energy loss
    - Calorimeter hits
    - Wire chambers
    - Hodoscope tiles
Ring trackers

```
g2MIGTRACE >> ls /g2MIGTRACE/rootStorage/
Command directory path : /g2MIGTRACE/rootStorage/

Guidance :
Root File Storage Management

Sub-directories :
   /g2MIGTRACE/rootStorage/g2UOM/ Unique Object Manager Query Commands
   /g2MIGTRACE/rootStorage/ringHits/ Ring hit monitor controls
Commands :
   storageStatus * Determine whether a Root file is created to store simulation results
   basename * Get/set the base filename: <basename> <datetime>_run#.root; omit arg for "get"
   outdir * Get/set the output directory; omit arg for "get"
   inflectorTrackerStatus * Get/set the inflector tracker enable state; omit arg for "get"
   ringTrackerStatus * Get/set the ring tracker enable state; omit arg for "get"
   ringHitStatus * Get/set the ring hit monitor enable state; omit arg for "get"
   caloHitStatus * Get/set the calorimeter hit monitor enable state; omit arg for "get"
g2MIGTRACE >>
```
Ring trackers

```
g2MIGTRACE >> /g2MIGTRACE/rootStorage/ringTrackerStatus
/g2MIGTRACE/rootStorage/ringTrackerStatus
The ring beam trackers are not enabled
```
```
g2MIGTRACE >> /g2MIGTRACE/rootStorage/ringTrackerStatus on
/g2MIGTRACE/rootStorage/ringTrackerStatus on
```
```
g2MIGTRACE >> /g2MIGTRACE/rootStorage/ringTrackerStatus
/g2MIGTRACE/rootStorage/ringTrackerStatus
The ring beam trackers are enabled
```
```
g2MIGTRACE >>
```
Ring Trackers

- `TTree trackerTree`
  - `std::vector<trackerRecord>`
- What's a `trackerRecord`?
/** Provides a Root storable class to record in-ring beam tracking data. Uses the data recorded by the trackerSD class. */

struct trackerRecord {

    /** Horizontal offset from the central orbit, mm. */
    Float_t rhat;
    /** Vertical offset from the central orbit, mm. */
    Float_t qhat;
    /** Global angular offset downstream of the arO/arcl11 boundary (NOT the inflection point!). */
    Float_t theta;
    /** Global event time, ns. */
    Float_t time;

    /** Total momentum, MeV. */
    Float_t p;
    /** Fraction of momentum along the rhat direction. */
    Float_t prhat;
    /** Fraction of momentum along the qhat direction. */
    Float_t pqhat;

    /** Current orbit number since injection. */
    Int_t turn;
    /** Unique volume ID of the current tracking volume; use with the g2UniqueObjectManager for volume identification. */
    Int_t volumeUID;
    /** Current track ID; used with the stored trackRecord for particle identification. */
    Int_t trackID;
};
The coordinate system

- In `trackerRecord`, we have a hybrid cylindrical/toroidal system
  - $r$ and $z$ should be obvious
- The angle ... well, not so much
  - The angle is defined as “downstream” from a convenient reference point
  - Almost all objects are located within vacuum chamber sections by relative measurements
  - Hence, the global angle is defined on the arc0/arc11 boundary ... slightly downstream of the nominal inflector aperture
To whit...

Inflector aperture position relative to 0 degree of arcSection 0
-5.587 degrees

0 degree of arcSection 0

Start of Q1 relative to 0 degree of arcSection 0
16.323 degrees
There's a pattern here...

- A somethingRecord is the unit of storage of a single particle Step in the Event
- somethingRecords are generally stored in a std::vector<somethingRecord>
- That vector is a leaf in the somethingTree
- Entries in parallel somethingTrees are all from the same simulation Event
/** Provides a Root storable class to record in-ring beam tracking data. Uses the data recorded by the trackerSD class. */

struct trackerRecord {

  /** Horizontal offset from the central orbit, mm. */
  Float_t rhat;
  /** Vertical offset from the central orbit, mm. */
  Float_t what;
  /** Global angular offset downstream of the arO/arc11 boundary (NOT the inflection point!). */
  Float_t theta;
  /** Global event time, ns. */
  Float_t time;

  /** Total momentum, MeV. */
  Float_t p;
  /** Fraction of momentum along the rhat direction. */
  Float_t prhat;
  /** Fraction of momentum along the what direction. */
  Float_t pvhat;

  /** Current orbit number since injection. */
  Int_t turn;
  /** Unique volume ID of the current tracking volume; use with the g2UniqueObjectManager for volume identification. */
  Int_t volumeUID;
  /** Current track ID; used with the stored trackRecord for particle identification. */
  Int_t trackID;
};
Volume identifiers

- Between runs, volumes can be moved/added/removed!
- At BeginOfRun (such as `/run/beamOn`), the geometry is *frozen* and *voxelized* ... from then on, it can't change until EndOfRun
- Root files are stored per run, hence, a reference database of all volumes is built and stored *without work on your part*!
  - But!!! Physical Volume Names must be *unique* for this scheme to work!
- The DB is called `g2UniqueObjectManager` and provides a useful set of services
class g2UniqueObjectManager : public TObject {
public:

/** Registers a volume with the UID store, mapping the physical volume pointer, \a ptr, to a unique integer ID, and associating that with the volume identity, it's \a name. */
bool add(void* ptr, std::string name);

/** Clears the UID store. */
void clear();

/** Looks up a volume name given its UID, \a uid. */
std::string lookup(ULong64_t uid) const;

/** Determines whether a given volume, \a uid, has a name that matches a given pattern, \a p. */
bool re_match(ULong64_t uid, std::string p) const;

/** Returns a list of all the volumes in the store whose names match the pattern \a p */
std::vector<std::string> re_match_names(std::string p) const;

/** Returns a list of the UIDs for volumes in teh store whose names match the pattern \a p */
std::vector<ULong64_t> re_match_uids(std::string) const;

/** Counts the number of entries in the UID store. */
int count() const;
};
User default configuration .g2MIGTRACE/default_user_config.migtrace not present.
SVN Build Version 232

******** WELCOME TO THE g2MIGTRACE SIMULATION ********

g2MIGTRACE >> /run/beamOn 1
/run/beamOn 1
Entries in UOM: 228

Muons injected : 1
Muons stored : 0
Capture Efficiency : (0 +/- 0)%

Elapsed time this run: 0.123990051sec
Unique Objects in Manager: 228

---
/** Provides a Root storable class to record in-ring beam tracking data. Uses the data recorded by the trackerSD class. */

struct trackerRecord {

    /** Horizontal offset from the central orbit, mm. */
    Float_t rhat;

    /** Vertical offset from the central orbit, mm. */
    Float_t vhat;

    /** Global angular offset downstream of the arO/arc11 boundary (NOT the inflection point!). */
    Float_t theta;

    /** Global event time, ns. */
    Float_t time;

    /** Total momentum, MeV. */
    Float_t p;

    /** Fraction of momentum along the rhat direction. */
    Float_t prhat;

    /** Fraction of momentum along the vhat direction. */
    Float_t pvhat;

    /** Current orbit number since injection. */
    Int_t turn;

    /** Unique volume ID of the current tracking volume; use with the g2UniqueObjectManager for volume identification. */
    Int_t volumeUID;

    /** Current track ID; used with the stored trackRecord for particle identification. */
    Int_t trackID;
};
Track Identifiers

- In every Event, every the birth of every particle is recorded
- `trackRecord` -> `std::vector<trackRecord>` -> `trackTree`
struct trackRecord {

    /** Usually the particle name. */
    std::string trackType;

    /** The Geant track ID of the current particle, which is stored in some other Record types. */
    Int_t trackID;

    /** The Geant track ID of this particle's parent. This equals itself for the primary. */
    Int_t parentTrackID;

    /** The current orbit number at particle birth. */
    Int_t turn;

    /** The current physical volume UID the track was born in. */
    Int_t volumeUID;

    /** The radial offset from the nominal storage orbit at the particle's birth. */
    Float_t rhat;

    /** The vertical offset from the nominal storage orbit at the particle's birth. */
    Float_t vhat;

    /** The azimuthal angle downstream from the global zero at the particle's birth. This is a few degrees downstream of the inflection point; see the documentation directory. */
    Float_t theta;

    /** The global time at the particle's birth. */
    Float_t time;

    /** The total momentum given the particle at birth. */
    Float_t p;

    /** The fraction of the particle's momentum which is oriented radially at birth. */
    Float_t pihat;

    /** The fraction of the particles's momentum which is oriented vertically at birth. */
    Float_t pvhat;
};
Doing something useful

- Enable what you want
- Run
- Write some analysis code
Writing some analysis code

- Currently, you'll have to do a manual build/link step on your analysis code ... the build system provides you no assistance.
  - You'll need to augment your include path so the headers in `g2MIGTRACE/trunk/include` are found.
  - You'll need to link against `$G4WORKDIR/tmp/$G4SYSTEM/libROOTRecords.so`.
  - Then you can open the TFile and analyze away!
- There's no code in the repository, but some can be provided on request.
For developers

- How it all works
- How to add output
How it all works

- The runtime component
  - rootStorageManager
  - g2UniqueObjectManager
  - Sensitive Detectors and the TTrees

- The build component
  - Dictionaries
  - File naming conventions
rootStorageManager

- A singleton responsible for all Root specific operations
  - Activate/Deactivate sensitive detectors associated with data types to be stored
  - Opening TFiles
  - Storing Metadata
  - Booking/branching/writing TTrees
  - Converting from Geant4 implementations to Root storable types
  - Writing and closing TFiles
g2UniqueObjectManager

- Bidirectionally maps the Physical Volume Name to a (much much!) shorter UUID (the 64bit linear address of the instantiation)
- UUIDs are stored in various places in lieu of Volume Names
- Provides lookup services
  - By name (regular expressions!)
  - By UUID
Sensitive Detectors and the Trees

- All step-by-step data collection is done within Sensitive Detectors
- SDs are Activated by the same code within the rootStorageManager that books the Trees
- They are also Deactivated if the trees aren't booked!
- Trees are sorted, translated, and written to the Root file by rootStorageManager when EventAction::EndOfEvent fires
SD Translation

- The Geant4 classes traffic in Geant4 classes (duh)
- The Root persistence framework traffics in Root persistable classes (duh)
- These are not the same code!
  - You can't easily store Geant4 data in Root classes
- `rootStorageManager` provides a translation layer ... you write a converter, and the manager does the rest
  - `trackerHit` -> `trackerRecord`
The build component

- Standard set of makefiles
  - DAGs! Learn to do it right!
- Persistable types must follow these rules to produce a working dictionary:
  - Class declaration in include/newRecord.rhh
  - Class definition in src/newRecord.rcc
    - Even if “trivial”/empty!
  - LinkDef header in include/newLinkDef.h
    - Again, even if empty ... which it probably shouldn't be
- New Sensitive Detector hit types should be called newHit.{hh,cc}, and a convert function should be written as in src/rootStorageManager.cc