GenTools, Generators in Gaudi

Brett Viren

Physics Department

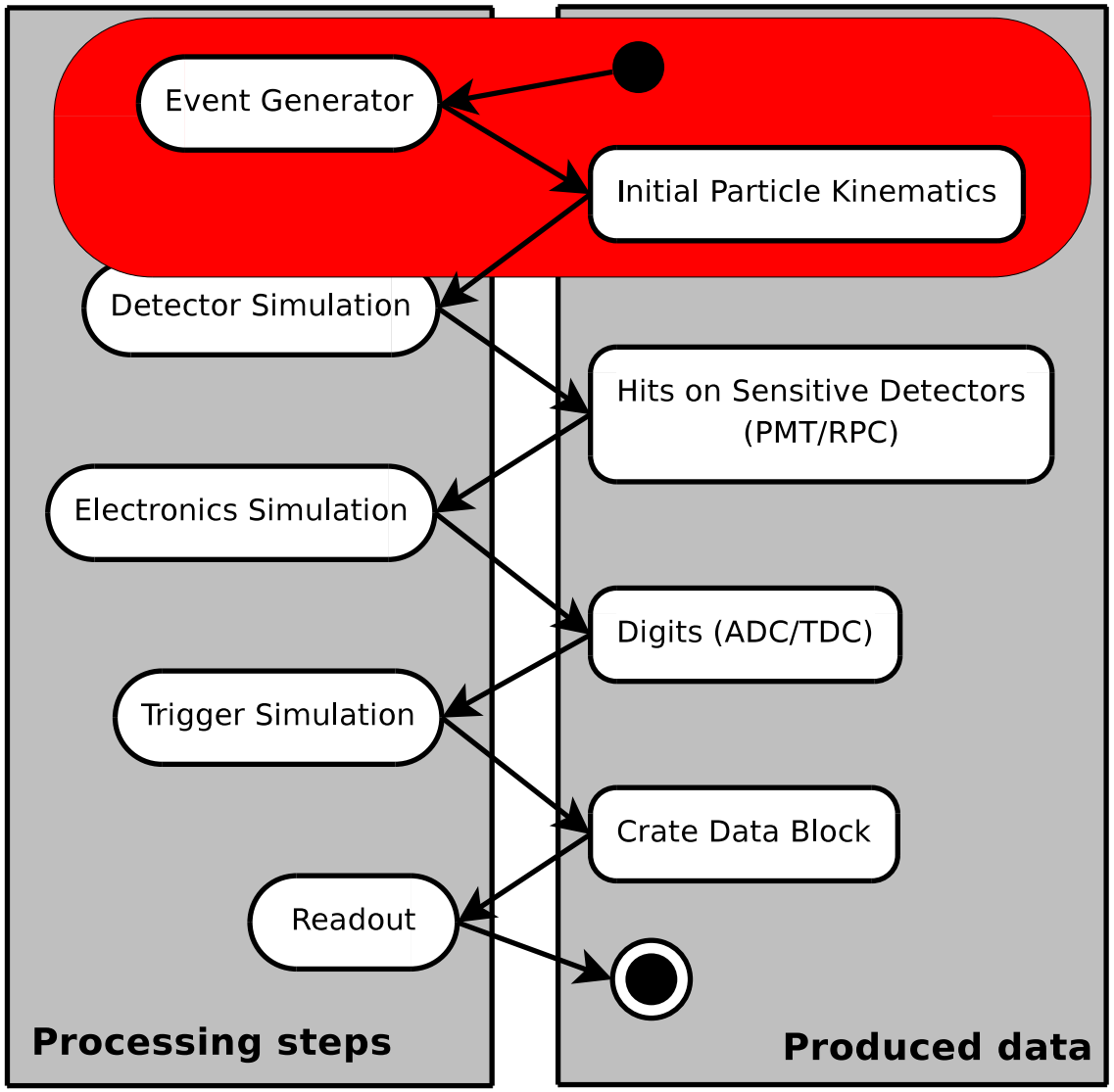


Daya Bay Simulation Phone, 2008/01/07

Contents

1. Context
2. Generator classes
3. Outputs
4. Job Configuration
5. Writing yor own Generator Tool
6. To do list

Just the Event Generator, (DetSim, etc, later)



The Generator class

- Lives in `Simulation/GenTools` package (`Gt` for short).
- A single, reusable `GtGenerator` algorithm
- Responsible for:
  - Setting up TES output object. By default uses: `/Event/Gen/HepMCEvents`, but it is configurable.
  - Using one or more "event mutator" tools to completely or partially build the event. These tools do the real work.
  - Storing the final result in the TES
- Possible to generate multiple events with `GaudiSequencer` + multiple `GtGenerator` instances (but this doesn't get us our "15 minutes" yet).

Generator "event mutator" Tools

Existing tools:

`GtHepEvtGenTool` : generate events from a HEPEvt source.

- Source is a file or an executable, can load more than one
- Fully rewritten (not G4dyb's version) no Geant4 needed.

`GtPositionerTool` : place an event in a volume

- `FullVolume` - choose uniform random in entire volue
- `AvoidDaugthers` - as above, but don't fill space ocupied by daughters.
- `Surface` - uniform on surface (not yet implemented, volunteers?)

`GtTimeratorTool` : place an event in time

- Given a rate, select a time based on exponential dist.

`GtGuneratorTool` : classic particle "gun"

- Still needs implementation (volunteers?).

Generator output

- One generator produces one `HepMC::GenEvent`
  - Comunity standard for generator information
  - `GenEvent` holds `HepMC::GenVertex`'s which hold `HepMC::GenParticle`'s
- Default TES storage: `/Event/Gen/HepMCEvents`
- Of type: `DayaBay::HepMCEvents` which is a collection of `DayaBay::HepMCEvent` each which hold one `HepMC::GenEvent`.
- The `DayaBay::` classes are part of our "data model".

These are then used as input for the Detector Simulation (DetSim), but let's leave that for a later meeting (so I can finish it!).

Job Configuration

```
toolSvc = app.toolSvc()
hepevt = toolSvc.create("GtHepEvtGenTool")
hepevt.HepEvtDataSource = "/path/to/Generator.exe -n 10"
app.TopAlg += [ "GtGenerator/Gen", "GtHepMCDumper/Dumper" ]
gen = app.algorithm("Gen")
gen.GenTools = [ "GtHepEvtGenTool" ]
gen.GenName = "My Special Generator"
gen.GenEventLoc = "/Event/Gen/MySpecialHepMCEvents"
```

See GenTools/share/test-GeneratorAlg.py for full example.

How to write your own GenTool (for reference)

Best to add it in your own package. If you think it belongs in `GenTools/`, talk to me.

1. Inherit from:
   `public GaudiTool, public IHepMCEventMutator`
2. Call this in the constructor:
   `declareInterface<IHepMCEventMutator>(this);`
3. Provide method:
   `virtual StatusCode mutate(HepMC::GenEvent& event)`
4. Add to `MyPackage/src/MyPackage_entries.cc` (or your own equivalent, see `GenTools/src/GenTools_entries.cc` for full example):
   - `#include<MyGenTool.h>`
   - `DECLARE_TOOL_FACTORY(MyGenTool);`
   - `DECLARE_TOOL(MyGenTool);`

See `GenTools/src/GtTimerator.*` for a most simple example.

Work that needs doing

- The traditional particle "gun" needs writing.
  (LHCb's `Gen/ParticleGuns/` is ripe for stealing.)
- Implement the "Surface" event placement,
  (when someone actually needs it).
- Further integration of the old style executable generators desired:
  - Would like to remove the requirement to match the # of generated events to the # of processed events.
  - Remove requirement of HEPEvt (still can have it if useful)
- Any user testing is appreciated, and ready for it now.

Any Volunteers?

A last one, which I think is only for me:

- Write Stage/Process classes to implement just-in-time processing needed for D&B's "15 Minutes Of Data" design. This will re-use the generator tools.