Planning Geant4's next Major Release

Geant4 Collaboration Workshop

Ben Morgan (The University of Warwick)

Rationale for starting this discussion

- If lifetimes of Geant4's major versions are a guide, we might expect the next one in 3-4 years from now
  - *About to release 11.3, last minor releases of 9, 10 were 9.6 and 10.7*
- LHC's Long Shutdown 3 (2026-28) provides a possible window for major updates without impacting its experiments production needs
  - ***That's HEP, and need to consider all the other user communities here***
- So why start thinking about it **now**?
  - *Major releases are **opportunity for breaking changes to interfaces/behaviour to improve Geant4's technical/physics performance, robustness, usability…***
  - *… but that requires careful **preparation** (what we want to do) and **prioritization** (what we can do, given available FTE).*
  - *Also needs time to gather feedback from stakeholders/users on requirements, limitations they may have that warrant or defer a major release*
- **Nominal 1yr integration period for releases** means solutions need to have been prototyped/tested in advance of the year the new major release is prepared

Where/How to start?

- Use **this week** to start thinking/discussing amongst ourselves
  - *Limitations on timescale for the next major release from stakeholders/users?*
  - ***What ideas do we have for interface/implementation changes we could make to improve Geant4's physics/technical performance and usability, given the freedom offered by a major release?***
  - *The good, the bad, the ugly… don't be afraid to be radical here, but need to note down* ***cost/benefit to us and users***
- **CodiMD Doc for you to write ideas, comments, down:**
  - ***https://codimd.web.cern.ch/fwCUqelaTnGl-PvbX2cYUA***
- **Nothing more than information gathering** to see what ideas there are, how much interest there is, and what next steps might be, **if any**, over the coming months/years
  - ***Emphasize that development focus still very much on consolidating release 11!***
  - ***Equally, might identify things we can start implementing now, without breaking interfaces!***
- Simply to motivate discussion and illustrate what *could* be considered for a major release, will outline some ***personal*** ideas for breaking changes

Personal idea: Ownership of Resources 1

- Not always clear who owns an instance created through new:
  - *Raw `new` is confusing to users (and goes against all modern C++ teaching)*
  - *Even if pointer held in local variable, not always a corresponding `delete`*
- Raw pointers passed around a lot and can be extremely confusing to trace who owns (i.e. has responsibility of deleting) what.
  - *If I'm returned a raw pointer, do I own it?*
  - *If I pass a raw pointer into something, do I retain or hand over ownership?*
- Use of raw `new`/`delete` for class data members where other solutions may fit better:
  - *`std::optional` or `operator bool()` for "may not be initialized/available"?*
  - *`std::unique_ptr` for collections of owned pointers, or additional support classes for collections?*

Personal idea: Ownership of Resources 2

- C++11/17 provide smart pointers to help here, so can we use them more extensively internally and in public interfaces to clarify this?

```
// e.g. instead of
G4Foo* SomeFunction();

// clarify that caller gets ownership
std::unique_ptr<Foo> SomeFunction()
```

- If an object must be created through new, should we make constructors private, and have a factory function instead?

```
// e.g. instead of
new G4SomethingStored(“foo”);

// hide new in factory function
G4SomethingFactory::Make(“foo”);
auto x = G4SomethingFactory::Make(“bar”);
```

Personal idea: Reduce Global Statics/Singletons

- Though Geant4 is a library, actually difficult, if not impossible, to setup, teardown, and re-setup new “experiment” inside same application
  - *As outlined here: https://gitlab.cern.ch/geant4/geant4-dev/-/issues/140*
  - *Primarily due to global statics/singletons and their setup/teardown, or lack thereof*
- Removing them entirely would be a huge redesign, so impractical now(*), but maybe some mitigations/improvements possible:
  - *Are there any cases where a singleton or otherwise global isn’t really needed now, or would be easy to de-singleton it?*
  - *Could we improve/provide explicit “teardown” functions/interfaces so we don’t have to rely entirely on static deletion?*
- (*) but maybe review use and see how they might be in future…

Personal idea: Multithreading and Allocators

- Still retain "`GEANT4_BUILD_MULTITHREADED`" configuration option, so could we now remove this and just always build with MT support?
  - *Potential impact to sequential performance, but if so how much?*
  - *A cost/benefit exercise in reducing build complexity against runtime performance*
- Whether we could remove "classic" MT mode, only use Tasking.
  - *Again, cost/benefit in performance, long term support (of Tasking), ease of use*
- Allocators and Thread-Local Storage
  - *Cannot deallocate an object on a different thread to that where it was allocated*
  - *E.g. if we wanted to make more use of Tasking, can't guarantee which thread a Task runs on*
  - *Look to be technical solutions out there, but requires more research and testing.*

Personal idea: Learning from R&D Projects

- R&D projects are just that, but even if still in progress, are there things we could learn from that might be applied more broadly?
- From the G4HepEM and GPU R&D projects:
  - *Better structuring of data (e.g. cross-sections) for efficient memory access?*
  - *Structure of stepping loop?*
- From Python/Julia bindings:
  - *What, exactly, in Geant4 are user interfaces, and what are toolkit-internal.*
    - *This is a question about what classes/functions to bind/expose.*
  - *Backporting/feedback on user interface usability/convenience*

Personal idea: Doxygen-ation of primary classes

- Something we can actually be doing now (modulo defining the style of comments)!
  - *Partially a repayment of technical debt that we've accumulated over the years*
- Basically, consistently format/define "docstrings" that describe a class/function/etc's purpose, inputs, outputs, pre/postconditions, e.g.

```
// See G4String.hh
/// @brief Return lowercased copy of string
/// @param[in] str the string to lowercase
/// @return lowercased copy of `str`
inline G4String to_lower_copy(G4String str);
```

More details on the Doxygen site: https://www.doxygen.nl/manual/docblocks.html

- "It's too much work…": ***no***, most descriptions are there, just needs migration. Which can be done step by step (hence can start doing now)
- See, e.g https://root.cern/doc/v632/classTTree.html for what's generated

Summary, and what next?

- So those are my thoughts… (criticism welcome!)
- **… what are yours?**
- Use this week to discuss/argue about ideas and timescales for a next major release of Geant4
  - *What blockers/limits are then in time from our stakeholders and users?*
  - *What ideas, if any, are there that would require a major release to implement?*
  - *What costs/benefits to users do you think there are?*
  - *Etc…*
- **Use the CodiMD doc to note down your thoughts/objections** (but put your name against them so they can be discussed!):
  - *See where we are at the end of this week, what next steps there might be (and there might be none at this point!).*