Outlook

Identifying a physical volume uniquely - to create a 64-bit identifier for sensitive touchables

From John Apostolakis <John.Apostolakis@cern.ch>

Date Wed 1/8/2025 18:36

To Markus Frank <Markus.Frank@cern.ch>

Cc Andrei Gheata <Andrei.Gheata@cern.ch>; Gabriele Cosmo <Gabriele.Cosmo@cern.ch>; Lorenzo Pezzotti <lorenzo.pezzotti@cern.ch>

Dear Markus,

Thank you for taking the time to discuss with me this morning.

I write as a follow up to summarise my understanding from our discussion, document our progress and identify what may remain to resolve.

Background:
- In DD4HEP you create a unique identifier for each sensitive touchable (ie a sensitive physical instance of a logical volume + its full geometry hierarchy),
- This identifier is stored in the hits in the simulation and read by the reconstruction to assign them to the correct 'sensitive touchable' in the hierarchy
- The unique identifier is created by combining a bit mask for each physical volume in the hierarchy of each potential sub-detector.
- You allow the user to decide the value of the bit pattern for the set of physical volume at each level.
- The sub-detector is given 8 bits, and you decide per sub-detector how many bits to allocate for each level of its volume hierarchy to this identifier.
- Currently you have a map from a vector of physical volume pointers to this 64-bit identifier
- You (originally) requested a way to associate a 64bit user defined value with a physical volume. This value could then be used to derive your full identifier - likely by using an XOR operation.

Analysis:
In our discussion it became clear that only your application could create the bit value which you sought - as the bit strings could be created by users for the physical volume at each level.
So your requirement indeed is to store a unique bit-pattern for each physical volume - which can then be XOR-ed to obtain your final 64-bit id/pattern.

The interface of the (virtual) physical volume G4PVPhysicalVolume Abstract Base Class already includes the method

```
inline G4int GetInstanceID();
```

which returns zero or a positive integer, unique to each physical volume constructed.

I note that the InstanceID method has existing since Geant4 version 10.0, as it was part of our multi-threading port of Geant4. So any solution that depends on it will be portable between Geant4 releases since 10.0 - ie and the full 10.x and 11.x series.

Potential Solution:
As we discussed, the GetInstanceID method seems to provide the key ingredient required to construct a new implementation for creating the 64-bit identifier. It can either be used to recreate this identifier by XORing its value level by level, or a caching mechanism could be used up to a certain level before proceeding to add the remaining levels.

— This concludes the summary of the key elements of our discussions, as I recall them.

An issue / problem

Looking at G4VPhysicalVolume type (header) reminded me of an inconvenient fact - that only G4PVPlacement volumes are uniquely identified by this instance ID. Multiple slices in a single 'Replica' (or 'Division') will share this identifier, as they are represented by a single physical volume in memory. Similarly all instances of a 'physical volume Parameterisation' share the same identifier.

Open question(s):
- Do you allow a user to create a 'Replica' or 'Division' which slices the original volume into separate sub-volumes? If so, they share the same physical volume pointer (a G4PVReplica or G4PVDivision) and as such would share the same Instance ID. So a separate sub-solution would be required for these types. It is possible to assertain whether a physical volume is one of these types - but likely you already know this during the construction of your geometry.

— Potential extension / solution

I can imagine a simple extension to the InstanceID which could be used to identify every physical volume uniquely. For each repetetive volume (replica, division, parameterised volume) it would create a new id combining an offset plus its replica-number. The offsets could be pre-calculated at the start of the application to provide enough 'space' to ensure the new / revised identifier is unique, and that either no gaps are created (or that padding is minimal.)

I hope that I have summarised our discussions well, and provided the means to start a prototype using this 'hidden' part of the Geant4 implementation.

Best regards,
John Apostolakis