art - Feature #1214

Ptr to Run or SubRun product

04/16/2011 09:35 PM - Rob Kutschke

Status:	Assigned	**Start date:**	07/25/2013
Priority:	Normal	**Due date:**	01/01/2014
Assignee:	Kyle Knoepfel	**% Done:**	100%
Category:	Navigation	**Estimated time:**	0.00 hour
Target version:		**Spent time:**	0.00 hour
Scope:		**SSI Package:**	art
Experiment:			

Description

Is it possible to use Ptr to point from an object in an Event data product to an object that is in a Run data product or a SubRun data product? If not, I would like to request that this feature be added (either to Ptr or, if not practical, to a new class template). Our SimParticle object needs to know the volume in which it was created and the volume in which it died. Currently we maintain an data product with volume names inthe Run product space and the SimParticle class holds two ints that are indices into the volume name product.

Subtasks:

Milestone # 4432: Carry out analysis to determine the work necessary to implement Ptr f...	**Closed**
Feature # 4433: Actually implement Ptr for Run & SubRun	**Closed**

Related issues:

Related to art - Feature #1918: Thinking ahead to event mixing and art::Ptr t...	**Accepted**	**09/24/2011**	**01/01/2014**
Blocked by art - Feature #16602: Allow multiple files with inconsistent proce...	**Closed**	**05/19/2017**	

History

#1 - 04/18/2011 09:46 AM - Marc Paterno

- Category set to Navigation

- Status changed from New to Feedback

Currently, Ptr<> objects stored in an Event are implicitly known to refer to an item the same Event, Ptr<> objects stored in a SubRun are implicitly known to refer to an item in the same SubRun, *etc.* We would need an extension of the design to allow more flexibility.

Two possibilities come immediately to mind:

1. At the cost of a slight increase in size on disk, we could add a persistent data member to Ptr<T> that indicates what it points "into"; an Event, SubRun, or Run.
2. At no cost in size, but a slight increase in number of types the user needs to know about, we could make there be three Ptr<T>-like templates, one that points to objects in an Event, another that points into objects in a SubRun, and a third that points into objects in a Run.

Unless there is a need to have collections of Ptr<T> objects which contain a mix of the three "types" of Ptr<T> objects, I would prefer solution 2; it doesn't cost in disk space size, and it makes it harder to make a mistake in associating items.

#2 - 04/18/2011 10:31 AM - Rob Kutschke

Notes from a conversation with Marc.

I prefer Marc's solution 2. We talked about adding a second argument to the template that identifies the target as InEvent/InRun/InSubRun, with a default to InEvent. So long as no one else is using Ptr to point SubRun to SubRun or Run to Run we will not break existing code.

We discussed the following: do we ever want to make a collection of Ptr types that includes some objects that are Ptr to a target in an Event and others that are Ptrs to a target in a Run or SubRun. The answer is no. I believe that all such attempt will be logic errors on the part of the coder.

We sketched what an event-data class might look like when it includes a Ptr to an object in the run space.

```
class T;

class Thing{
public:
Ptr<T,InRun> t() const { return t_; }
private:
Ptr<T,InRun>  t_;
```

};

t_ needs to be initialized as

t_(handleToACollection, index).

The designer of the data class Thing needs to know about the detailed type of t_ but users never do. User will normally use it as:

Thing x;
T const& t = *x.t();

They only need to know the type of the object that they are trying to get.

Marc wants the dereference out of the class Thing so that there is weaker coupling between Thing and Event. I agree.

There is a safety issue that this pattern exposes. If a user intended to construct t_ using a handle to an object in Run space, but accidentally passed a handle to an object in Event or SubRun space, then the creation of t_ will work. When you read back the event and try to follow t_ it will either give a run time error or will produce an incorrect result. To get an incorrect result there will need to be an object in the Run space that has both the same type and ProductId as the object in the event space; while this is not impossible it seems it will be rare. If we decide to do so, we can close this loophole, at a later date, by making the handle classes aware of Event/SubRun/Run

#3 - 04/20/2011 11:24 AM - Rob Kutschke

Can this change be done in such a way that NOvA can continue to read their old files?

#4 - 04/20/2011 11:31 AM - Marc Paterno

If backward compatibility with old data files is not an issue, it will be easier to produce efficient code. If backward compatibility is necessary, the solution I prefer would be to create a translation program that turns old-format files into new-format files.

If such a translation program is not acceptable, I believe we can make the schema evolution features of Root deal with this change. The code becomes more complex and more work probably needs to be done at run-time to deal with the fact that conversion might be needed; this may make the code less efficient even when reading new-format data. I would hope the backward-compatibility code would be a temporary feature, and that after a few releases the code could go away. If the old data files need to be readable forever, then the code would never go away.

#5 - 04/25/2011 06:03 PM - Christopher Green

Rob, while researching issue #1198 it has become apparent that Ptr to a a subrun or run product cannot work even within that run or subrun, never mind from event->subrun and friends. We would need to overhaul the metadata system in order to make this work. We would like to do this anyway for a whole host of reasons but it has not so far been necessary to implement an experiment-requested feature. If you wish us to do this, we can certainly do so but it would have to be formally requested.

#6 - 04/25/2011 07:22 PM - Rob Kutschke

We have one real use case for event->Run. So in the long run I really do want this feature. The question now becomes when do we do this and how do we deal with file format compatibility issues?

Here is how we use it now. Today our Run contains a std::vector<PhysicalVolumeInfo>. Each PhysicalVolumeInfo object gives the name and copy number of a G4 volume. Our SimParticle objects contain two integer indices that indicate the physical volumes in which particles were born and in which they died.

We can keep using this system for some time but it would be nice to plan now to avoid future issues about backwards compatibility of file formats. Does the following work? In the SimParticleClass, I can change my data member ints to Ptr<PhysicalVolumeInfo>. Then I can provide an accessor that mimics the current behaviour and returns the offset part of the Ptr as an int. The accessor will have the same name as the current accessor. In this way no code changes today but the layout on disk changes. This is not a big deal since I am making other non-backwards-compatible changes with the transition to art. When the Ptr starts working properly we can still read files created between now and then; at that time both the old and new style accessors will work. Then we can deprecate the old style accessor that returns an int and eventually delete it.

#7 - 05/06/2011 11:26 AM - Christopher Green

- Status changed from Feedback to Accepted

- Priority changed from Normal to Low

I think the best advice is to continue to use your offset; and when we have an implementation we should be able to use schema evolution to upgrade your product.

#8 - 12/09/2011 04:09 PM - Walter E Brown

- Priority changed from Low to Normal

#9 - 04/17/2015 10:22 AM - Kyle Knoepfel

- Target version set to 521

#10 - 06/15/2017 08:29 AM - Kyle Knoepfel

- Blocked by Feature #16602: Allow multiple files with inconsistent process histories added

#11 - 06/19/2017 02:31 PM - Kyle Knoepfel

- Status changed from Accepted to Assigned

- Assignee set to Kyle Knoepfel

- Target version deleted (521)

- SSI Package art added

The work done to resolve issue #16602 allows for art::Ptrs that refer to products *within their data containment level*. In other words, it is now possible to have:

- Event-level Ptrs to event-level products
- SubRun-level Ptrs to subrun-level products
- Run-level Ptrs to run-level products
- Results-level Ptrs to results-level products

It is not yet possible to have art::Ptrs that refer to products corresponding to a level that contains it (e.g. an Event art::Ptr that refers to a SubRun product). Is support for such a Ptr still desired/necessary?

#12 - 06/19/2017 03:24 PM - Rob Kutschke

Mu2e would still like the ability of an event level product to point to a SubRun or Run level product.

We have a particular use case, detailed below. We are open to other suggestions for how to solve the problem.

The use case is that we run Simulations in stages. Often we update the geometry information between stages (one of the reasons for staging is to allow us to simulate the expensive early stages during which time we stop particles entering restricted volumes that will be defined at a later stage; this permits us, for example, to start the expensive step of simulating protons on target while we are still working on defining the detector; it also permits us to consider variations on the detector design without the need for re-running the expensive earlier stages).

We store geometry information for each stage in a subrun product; the product is a collection type in which the stage number is the access key. At present one has to know the input tag of the subrun product, know how to get the subrun product from the event etc. With an Ptr into the subrun product the knowledge required to correctly access the geometry is reduced. This is really an easy-of-use optimization but we believe that it would be a valuable one to have.

#13 - 07/05/2017 05:41 PM - Christopher Backhouse

To do the equivalent on NOvA we have a service that looks in the Run at beginRun, and stores the geometry it finds there for later access. Of course, this doesn't really handle the case where there are multiple geometries stored there and you want to know which to use, unless the rule is always "most recent wins".