art - Feature #5474

Nicer syntax for iterating over std::Handle<std::vector<...

02/18/2014 02:11 PM - Christopher Backhouse

Status:	Closed	**Start date:**	02/18/2014
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	Navigation	**Estimated time:**	0.00 hour
Target version:		**Spent time:**	0.00 hour
Scope:	Internal	**SSI Package:**	art
Experiment:	NOvA		

Description

Currently, to iterate over an art::Handle<std::vector<DataProduct>> you have to do something like:

```
for(unsigned int i = 0; i < prods->size(); ++i){
  art::Ptr<DataProduct> prod(prods, i);
  ...
}
```

It would be nice to be able to do:

```
for(art::Ptr<DataProduct> prod: prods){
  ...
}
```

Currently you can do:

```
for(const DataProduct& prod: *prods){
  ...
}
```

but in some cases it's necessary to have the product as an art::Ptr inside the loop.

History

#1 - 02/18/2014 02:59 PM - Christopher Green

- Description updated

- Category set to Navigation

- Status changed from New to Feedback

- Assignee set to Christopher Backhouse

- Experiment NOvA added

- Experiment deleted (-)

Unfortunately, given the relationship between a collection and a Ptr into that collection, something like this is always going to be required, with its concomitant work. There is an alternative way which avoids having to create Ptr individually, but it's still a formalism:

```
art::View<X> v;
e.getView(tag, v);
art::PtrVector pv;
v.fill(pv);
for (auto const & p : pv) {
  ...
}
```

It's conceivable that we could update the interface so that this works:

```
for (auto const & p : e.getView<X>(tag).asPtrs()) {
  ...
}
```

We could also conceivably hide this work by providing a converting constructor for PtrVector<X>(HANDLE<SEQ<X>>), although it would involve some work to do this generally such that it works for all applicable cases.

My feeling however is that this would be likely to (further) confuse the Event interface and hide the fact that this is only an efficient thing to do if you intend to refer to every object in the View as a Ptr. It is in all cases less efficient than the method you are using already due to the creation of the View and PtrVector intermediate containers.

I believe the best solution is actually the one you already have since you can avoid manufacturing Ptrs that you don't need based on (*e.g.*) the actual object. Still, if you think this is a useful feature to have we can ask the stakeholders if this is a reasonable addition and where it would fit in their priority list.

#2 - 02/18/2014 04:24 PM - Christopher Backhouse

Do you really need the intermediate container?

Can't Handle get begin() and end() methods that return an iterator that consists of the index and a reference to the Handle, and when dereferenced constructs the Ptr? That gives exactly the same actions as my first snippet, but with the syntax of the second.

I'll admit my template wizardry isn't entirely up to scratch, but that's what I was imagining.

Obviously this isn't a top priority, but I do think it would be a nice reduction in the verbosity of our code (this loop, written in my first form, is extremely common).

#3 - 02/19/2014 05:39 PM - Christopher Green

- Assignee deleted (Christopher Backhouse)

The Handle::begin() and end() functions would make sense *only* for containers. It would mean specializing the class template for particular, "blessed" container types -- fiddly to get right, tricky to extend and possibly confusing to users.

"Fake" iterators that do work as they are de-referenced are also fiddly to get right and generally lead to more problems than they solve.

Unfortunately, what you're suggesting here is at least a couple days of work to save one line in each place the construct is used. We can certainly leave the issue in the queue if you wish, but I don't expect the work to be scheduled for quite some time.

#4 - 04/17/2015 10:24 AM - Kyle Knoepfel

- Target version set to 521

#5 - 12/05/2016 11:38 AM - Kyle Knoepfel

- Status changed from Feedback to Closed

- SSI Package art added

- SSI Package deleted ()

A similar usability enhancement is being developed within LArSoft. Once that feature has stabilized, we plan for it to migrate to art, for the benefit of all *art* users.

#6 - 10/23/2017 12:12 PM - Kyle Knoepfel

- Target version deleted (521)