New *art* feature: extended access to associations with metadata

Gianluca Petrillo

Fermi National Accelerator Laboratory

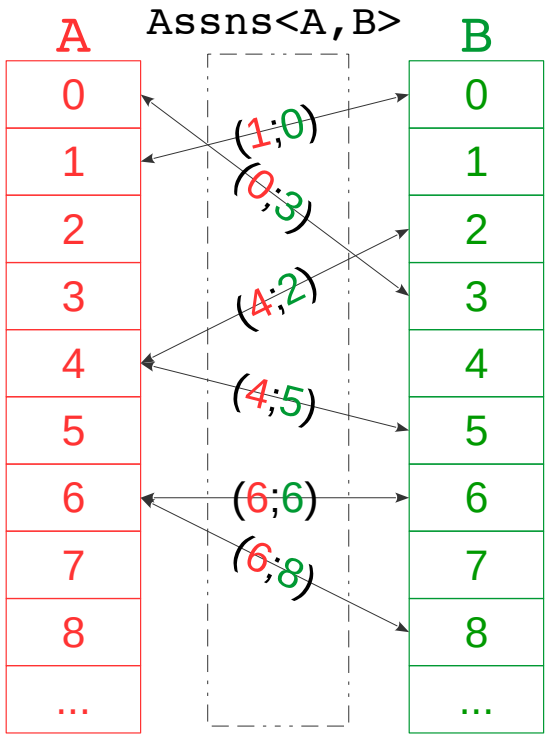
LArSoft Coordinators' Meeting, December 6th , 2016

Abstract

- a new *art* feature has been request by LArSoft and is being implemented
- there are decisions to be taken about how the feature behaves in some particular cases
- *art* team has requested feedback
- this talks describes the feature and elicits feedback to be forwarded

Reminder: what are associations, what is metadata

- associations relate elements of collections of two different types: `art::Assns<A, B>` relates elements of type `A` and elements of type `B`
- each relation can be supplied with additional data: `art::Assns<A, B, D>` adds to each relation between `A` and `B` data of type `D` (content of `D` is referred to as *metadata*)



- two ways to access them:
  1. directly: `event.getValidHandle<art::Assns<A, B, D>>(inputTag);`
  2. by query objects:

     `art::FindMany<B, D>(collectionOfA, event, inputTag);`
- products of type `art::Assns<A, B>` and `art::Assns<A, B, D>` have been formally unrelated: no way to get one in place of the other

Another reminder: data product identification

A *art* data product is identified by four elements:

ClassName_ModuleLabel_InstanceName_ProcessName

ClassName is a C++ class (e.g. `std::vector<recob::Track>`)
ModuleLabel is the name of the instance of the producer (not the producer class!) (e.g. "largeant" — not LArG4)
InstanceName is an optional name assigned by the producer
ProcessName is the name of the running *art* process (e.g. "DetSim")

Users can select a data product via FHiCL configuration by an *input tag*

ModuleLabel_InstanceName_ProcessName

- no run-time control on which class *art* loads: it's coded in producer
- if no instance name is specified, *art* looks for a empty name; in LArSoft experiments, it's almost never specified
- if no process name is specified, the most recent is used; I've never seen it specified

Yet another reminder: *art* data products lookup

art performs the lookup of a data product process by process, starting from the most recent one:

1. consider the products from the current *art* process (e.g., "Reco2"):
    1. if there is exactly one match, choose it and be done
    2. (currently multiple matches are not possible)
    3. otherwise, go to the process preceding this one in time
2. consider products from the previous *art* process (e.g., "Reco1"):
    1. if there is exactly one match, choose it and be done
    2. (currently multiple matches are not possible)
    3. otherwise, go to the process preceding this one in time (e.g., "DetSim")
3. ...
4. if no candidate was found anywhere, throw an exception (`ProductNotFound`)

An issue with usage and metadata

The scenario:

- an algorithm needs to know the hits in a track:

  `art::Assns<recob::Track, recob::Hit>`

- a tracker module wants to save information on each hit (residuals from the track, index of the node, *ds*...):

  `art::Assns<recob::Track, recob::Hit, reco::AlgoData>`

The issue:

- the algorithm asks for what it needs:

  `art::FindMany<recob::Hit> query(trackColl, event, inputTag);`

- associations `art::Assns<recob::Track, recob::Hit, reco::AlgoData>` do not match the request
- the tracker module is forced to "duplicate" the information by adding a association `art::Assns<recob::Track, recob::Hit>` without metadata describing the same relations as the other

art feature request #10539 allows an association with metadata (like `art::Assns<recob::Track, recob::Hit, reco::AlgoData>`) to fulfill a query for a simple association (like `art::Assns<recob::Track, recob::Hit>`).

The solution: a new feature

This schematics expresses the behaviour of *art* in answering queries:

<A, B, D>, L_I_Pass1	the associations available in the event so far, identified as the data type and its *complete* input tag. E.g., this one comes from: `produces<art::Assns<A, B, D>>("I");`
query <A, B>, "L_I" *old:* *throw!* *new:* <A, B, D>!	class type and the input tag in the query; it is typical in that it omits the process name. E.g., `art::FindMany<B> AtoB(collOfA, event, "L_I");` color-coded match by the current *art* color-coded match by *art* with feature #10539

The example above illustrates the purpose of the feature:

- a query asks for a `art::Assns<A, B>` with a certain input tag
- only a `art::Assns<A, B, D>` is available with that input tag
- #10539-featured *art* delivers that association (instead of complaining for a `ProductNotFound`)

Corner cases

These schematics illustrate some “corner cases” where ambiguities may arise:

<A, B, D>, L_I_Pass1
query <A, B>, "L_I" *old:* *throw!* *new:* <A, B, D>!

(feature target)

<A, B>, L_I_Pass1 <A, B>, L_I_Pass2
query <A, B>, "L_I" *old:* <A, B>! *new:* <A, B>!

<A, B, D>, L_I_Pass1 <A, B>, L_I_Pass2
query <A, B>, "L_I" *old:* <A, B>! *new:* <A, B>!

<A, B>, L_I_Pass1 <A, B, D>, L_I_Pass2
query <A, B>, "L_I" *old:* <A, B>! *new:* TBD

<A, B, D>, L_I_Pass1 <A, B>, L_I_Pass1
query <A, B>, "L_I" *old:* <A, B>! *new:* TBD

(current workaround)

<A, B, C>, L_I_Pass1 <A, B, D>, L_I_Pass1
query <A, B>, "L_I" *old:* *throw!* *new:* *throw!*

Policy: just don't go in the corner

- *art* team is looking for feedback on all those cases
- LArSoft proposal is a policy to avoid corner cases:

1. a producer will put multiple associations with the same instance name only when their common information is equivalent: that is, `art::Assns<A, B>` and `art::Assns<A, B, D>` will describe *exactly* the same $A \leftrightarrow B$ relations
2. if different relations are required, *instance names* must differ:

```
produces<art::Assns<recob::Track, recob::Hit>>(); // all hits
produces<art::Assns<recob::Track, recob::Hit, Info_t>>("onTrack");
```

This would not not solve the issues from products from different processes... I am not aware of this happening in LArSoft experiments.

This policy is in the form of an agreed *convention*.
It is not enforced by *art* code.
It is users' responsibility to make sure to comply.

Corner cases after the proposed policy

This policy "solves" one of the debated cases:

<A, B, D>, L_I_Pass1
query <A, B>, "L_I"
old: *throw!*
new: <A, B, D>!

⇒ unaffected by policy

(feature target)

<A, B>, L_I_Pass1 <A, B>, L_I_Pass2
query <A, B>, "L_I"
old: <A, B>!
new: <A, B>!

⇒ unaffected by policy

<A, B, D>, L_I_Pass1 <A, B>, L_I_Pass2
query <A, B>, "L_I"
old: <A, B>!
new: <A, B>!

⇒ unaffected by policy

<A, B>, L_I_Pass1 <A, B, D>, L_I_Pass2
query <A, B>, "L_I"
old: <A, B>!
new: TBD

⇒ unaffected by policy

<A, B, D>, L_I_Pass1 <A, B>, L_I_Pass1
query <A, B>, "L_I"
old: <A, B>!
new: TBD

⇒ it's the same!

(current workaround)

<A, B, C>, L_I_Pass1 <A, B, D>, L_I_Pass1
query <A, B>, "L_I"
old: *throw!*
new: *throw!*

⇒ (it was the same)

The other case still matters...

Feedback

Once again: *art* performs lookup of a data product process by process.

- that data product lookup policy explains the upper table and the "old" behaviour in the lower one.
- *art* would like feedback on it:
  1. intuition might suggest a <A, B> match
     - this requires reading *all input files* containing the event; this may hit performance hard when "fallback" files are used
     - in other words, if you have put your G4 data in a separate file, asking for track-hit association will have that file read, to check if there is any in there
  2. extrapolation of current *art* behaviour would predict <A, B, D>
     - the preferred one by *art* team (by far)

Tracks, L_I_Pass1 Tracks, L_I_Pass2
query Tracks? response: Tracks!

<A, B>, L_I_Pass1 <A, B, D>, L_I_Pass2
query <A, B>? *old:* <A, B>! *new:* TBD

Conclusions

- an awaited *art* feature request is going to be added soon
- some corner cases need to be regulated
- LArSoft proposes a policy that should remove most confusion and ambiguity
- still, feedback is required for one further case
  (experiments could consider policies to make this irrelevant too)