



Possible changes to *art*

Kyle J. Knoepfel

8 March 2018

SAM metadata

- *art* provides SAM metadata storage within the *art*/ROOT file.
- *art* provides the key-value entries; users are able to add additional key-value pairs.
- The data is extracted via `sam_metadata_dumper`, the output of which can be used when interacting with SAM.
- Recent issue report:

Every place we use `sam_metadata_dumper` we're having to filter its output because it generates keys and in some places values we cannot import.

SAM metadata

- *art* provides SAM metadata storage within the *art*/ROOT file.
- *art* provides the key-value entries; users are able to add additional key-value pairs.
- The data is extracted via `sam_metadata_dumper`, the output of which can be used when interacting with SAM.
- Recent issue report:

Every place we use `sam_metadata_dumper` we're having to filter its output because it generates keys and in some places values we cannot import.

- Stored SAM metadata was not necessarily meant to be mapped directly to what SAM requires.
- However, since its purpose is to provide easier interactions with SAM for *art* jobs, we are willing to adjust what we store to make it easier.

SAM metadata

- A proposal:
 - Make *art*'s stored SAM metadata align with SAM's specifications. *art*-specific keys and values will be stored in an “art” table.
 - For example, the supplied `first_event` and `last_event` values are integers according to SAM.
 - According to *art*, an event is uniquely specified as a triplet of run, subrun, and event numbers.
 - To avoid this conflict, the top-level `first_event` and `last_event` values will reflect the SAM requirement, and
 - Additional `art.first_event` and `art.last_event` values will be provided that reflect the *art* semantics.

SAM metadata

- A proposal:
 - Make *art*'s stored SAM metadata align with SAM's specifications. *art*-specific keys and values will be stored in an “art” table.
- Consequences of changing the schema:
 - Either `sam_metadata_dumper` will need to provide a translation from the old schema to the new one, or users will need to do so, or both.
- Thoughts?

art::Event::get

- Assume a product exists in the input file and has not yet been read into memory:

```
Handle<int> h;  
e.get(product_id, h);  
assert(h.isValid()); // fails
```

art::Event::get

- Assume a product exists in the input file and has not yet been read into memory:

```
Handle<int> h;  
e.get(product_id, h);  
assert(h.isValid()); // fails
```

```
Handle<int> h;  
e.getByLabel(product_id, h);  
assert(h.isValid()); // succeeds  
e.get(product_id, h);  
assert(h.isValid()); // succeeds
```

art::Event::get

- Assume a product exists in the input file and has not yet been read into memory:

```
Handle<int> h;  
e.get(product_id, h);  
assert(h.isValid()); // fails
```

```
Handle<int> h;  
e.getByLabel(product_id, h);  
assert(h.isValid()); // succeeds  
e.get(product_id, h);  
assert(h.isValid()); // succeeds
```

- Calling `art::Event::get` fills the handle *if* the product has already been read into memory. Otherwise, the handle remains invalid. This is not documented.
- Using `art::Event::get` is almost never the right thing to do. In fact, we would like to remove it if possible.
- One use case—a persisted reference to a data product is desired, but it is not an element in a container (i.e. `art::Ptr<T>` doesn't apply).
 - Solution: feature request for *art* to support persistable references to data products

art::Event::get

- Assume a product exists in the input file and has not yet been read into memory:

```
Handle<int> h;  
e.get(product_id, h);  
assert(h.isValid()); // fails
```

```
Handle<int> h;  
e.getByLabel(product_id, h);  
assert(h.isValid()); // succeeds  
e.get(product_id, h);  
assert(h.isValid()); // succeeds
```

Homework: does your experiment use `art::Event::get`, and if so, why?

- Calling `art::Event::get` is almost never the right thing to do. In fact, we would like to remove it if possible.
- One use case—a persisted reference to a data product is desired, but it is not an element in a container (i.e. `art::Ptr<T>` doesn't apply).
 - Solution: feature request for *art* to support persistable references to data products

Reducing severity of logged messages for open/close file

- There is a complaint that the severity of the logged messages indicating the opening and closing of a file is too high.
- The request is to reduce the severity to the info level.
 - The consequence is that these messages could be suppressed if the destinations are configured with a threshold higher than INFO.
- Any concerns about this?