art - Support #4324

File level data product storage

07/12/2013 10:10 AM - Brian Rebel

Status:	Closed	**Start date:**	
Priority:	High	**Due date:**	
Assignee:	Christopher Green	**% Done:**	100%
Category:	I/O	**Estimated time:**	0.00 hour
Target version:		**Spent time:**	0.75 hour
Scope:	Internal	**SSI Package:**	
Experiment:	NOvA		

Description

NOvA is trying to calibrate its Far Detector and would like to store the calibration constants in a data product at the File level. Currently the highest level available for storage is the Run level, however several calibration constants span multiple runs. Rather than writing many copies of the data product in the Run folder, we would like to have a File level folder in which to write the objects.

This is a high priority request as we are doing calibrations of the detector now.

History

#1 - 07/12/2013 11:37 AM - Christopher Green

- Category set to I/O

- Status changed from New to Feedback

After discussion in the stakeholders meeting, I'd like to schedule a requirements discussion, along the lines of the SAM meeting we had a couple of weeks ago. Given deadlines, vacations and SNOWMASS it's looking like it will have to be the second week of August. Is that acceptable?

#2 - 08/14/2013 04:15 PM - Christopher Green

- Status changed from Feedback to Accepted

- Start date deleted (07/12/2013)

- Scope set to Internal

- Experiment - added

- SSI Package art added

No-one replied to this ticket, so I will go ahead and create a doodle poll for a 2h requirements meeting to settle on the parameters for this implementation.

#3 - 08/16/2013 11:36 AM - Christopher Green

- Tracker changed from Feature to Bug

- Status changed from Accepted to Feedback

Based on further discussions with NOvA representatives at the stakeholders meeting it was decided that there be an internal NOvA discussion first, and a requirements meeting later should it be warranted. We await news of the outcome of NOvA's internal discussions.

#4 - 08/16/2013 11:36 AM - Christopher Green

- Tracker changed from Bug to Feature

#5 - 08/19/2013 04:11 PM - Christopher Green

- Status changed from Feedback to Assigned

- Assignee set to Christopher Green

- Experiment NOvA added

- Experiment deleted (-)

Brian forwarded this email:

From: Ruth Toner <rtoner@physics.harvard.edu>
Date: August 16, 2013, 10:36:00 PM CDT
To: Brian Rebel <brebel@fnal.gov>, Thomas Edward Coan <coan@mail.physics.smu.edu>
Subject: Drift plans and update

Hey Tom and Brian!

So as promised, here's a quick progress report on where things stand with the Drift calibration. I hope to have made more progress by next week, but part of that progress is contingent on some questions and concerns I have with how we're planning on doing the Drift Calibration.

Just to recap, the issue with the Drift right now is that we want to save the output of the DriftCorrection module in Art format, prior to uploading things to the database using a final (let's call it) DriftUploader analyzer module. This would be fine, but the current way that Art runs over and stores data makes this essentially trying to force a square peg into a round hole. My previous version of the code ran over all of the data in order and collated the Mean response per run for each channel. At the end of the code, it ran over the collated runs a first time to find the "good" runs, and then ran over them a second time to find the appropriate starts of new Validity Contexts and to calculate the corresponding drift correction. There are two issues with this:

A) Because of the way Art works (storing and accessing Data on a Run-to-Run basis), I need to do my Drift Correction analysis while I'm still in the process of adding Runs, rather than having access to the full time range of data. If I wait to the end, I'll be at "EndJob" and won't be able to write products to the output file anymore, unless the Art people have rewritten the framework to allow the writing of data products on a file-to-file basis (e.g., as a "Header"). This is not insurmountable - I can create an appropriately sized sliding buffer window of prior runs, and calculate the validity contexts and correction values on the fly. This requires careful thought and will get complicated quite quickly, however.

B) Because of the concerns in A, the way that the Drift Corrections constants get written to the Art output file will be really counterintuitive. Ideally, for each run, you'd like to write a Drift Response object. For each channel where a new Validity Context has been determined, a direct correction value X is recorded. If no new VC has been declared for that run, the correction value is -1. I still intend to use this basic framework. However, the Art Run the product "fResponse" is being written to, and the value "fResponse.Run" in the DriftResponse data product will not be the same, because by the time data quality and the drift correction has been calculated for run N, Art will have moved on ten or more runs into the future. You may also get numerous DriftResponse products for separate response.run values written to a single Art Run (let me know if this is confusing - it's been confusing to think about!). I also need to make sure that the constants have been calculated thoroughly for every channel before I write a data product, since the DriftResponse object contains information for all channels in a single Run (though I could get around this by writing a separate DriftResponse object for each channel). Long story short: whatever gets written to the Art output file will be a total mess, albeit one which can be sorted out by the DriftUpload module.

To be honest, using the moving window I describe in Part A is probably useful for memory management reasons anyway. And using a final "DriftUpload" module gives an opportunity to calculate the normalization (though this can just be done in a final EndRun job too frankly). However, both saving the Art data products for use by the DriftUpload module AND saving those same data products on a Run-to-Run basis is extremely messy. Do-able, but messy.

So just to summarize the above. I've been working on the DriftCorrection code, but the parameters requested have been a total pain to code up and have required some do-overs and very careful planning. I **can** do this, but please be advised that it might require more time and fine-tuning, and the output might be messy.

So basically: do you want me to keep trying for the above? I.e., writing my corrections run-by-run to an Art file, to be read by one final uploader mod? Or would you rather I try for something faster? (Writing to root, uploading to the database in EndJob instead of doing a new file…)

Let me know what you think - I figured it was time to ask for advice, rather than continuing to stew and be frustrated by this!

--Ruth

#6 - 08/19/2013 04:21 PM - Christopher Green

- Tracker changed from Feature to Support

- Status changed from Assigned to Resolved

- % Done changed from 0 to 100

Based on a discussion of Ruth's email with Brian and requirements arising therefrom, we think a better solution for this problem is in fact to write a DB table containing all the required information, including provenance, to the ROOT output file.

One way this can be done that does not require any changes to art is to have a service use the postCloseOutputFile() hook to open the ROOT file after closure and use an SQLite3Wrapper to access the DB in the file and create a table to contain the required information. This table's name should start with (eg) "nova." The callback should of course remember to properly close the DB and enclosing file after the work has been done.

This is not an ideal solution because the postCloseOutputFile() hook is meant to be nondestructive to allow for multiple hooks doing things like (e.g.) copying the file elsewhere. It is also not possible at this juncture to safely access the data from the current input file.

A future issue will be created to describe and implement the long-term solution to this question of DB access and it will be linked to this one. Ideally something created using the information herein would be upgraded to take advantages of the features provided by the implementation of said new issue.

Information and examples on creation and use of the SQLite3Wrapper may be found in source:art/Persistency/RootDB/SQLite3Wrapper.h and source:art/Framework/IO/Root/RootOutputFile.cc

#7 - 08/19/2013 04:21 PM - Christopher Green

- SSI Package added

- SSI Package deleted (art)

#8 - 03/03/2014 11:45 AM - Lynn Garren

- Status changed from Resolved to Closed