SQLITE Calibration Databases Update

Larsoft Coordination Meeting
Nov. 3, 2020

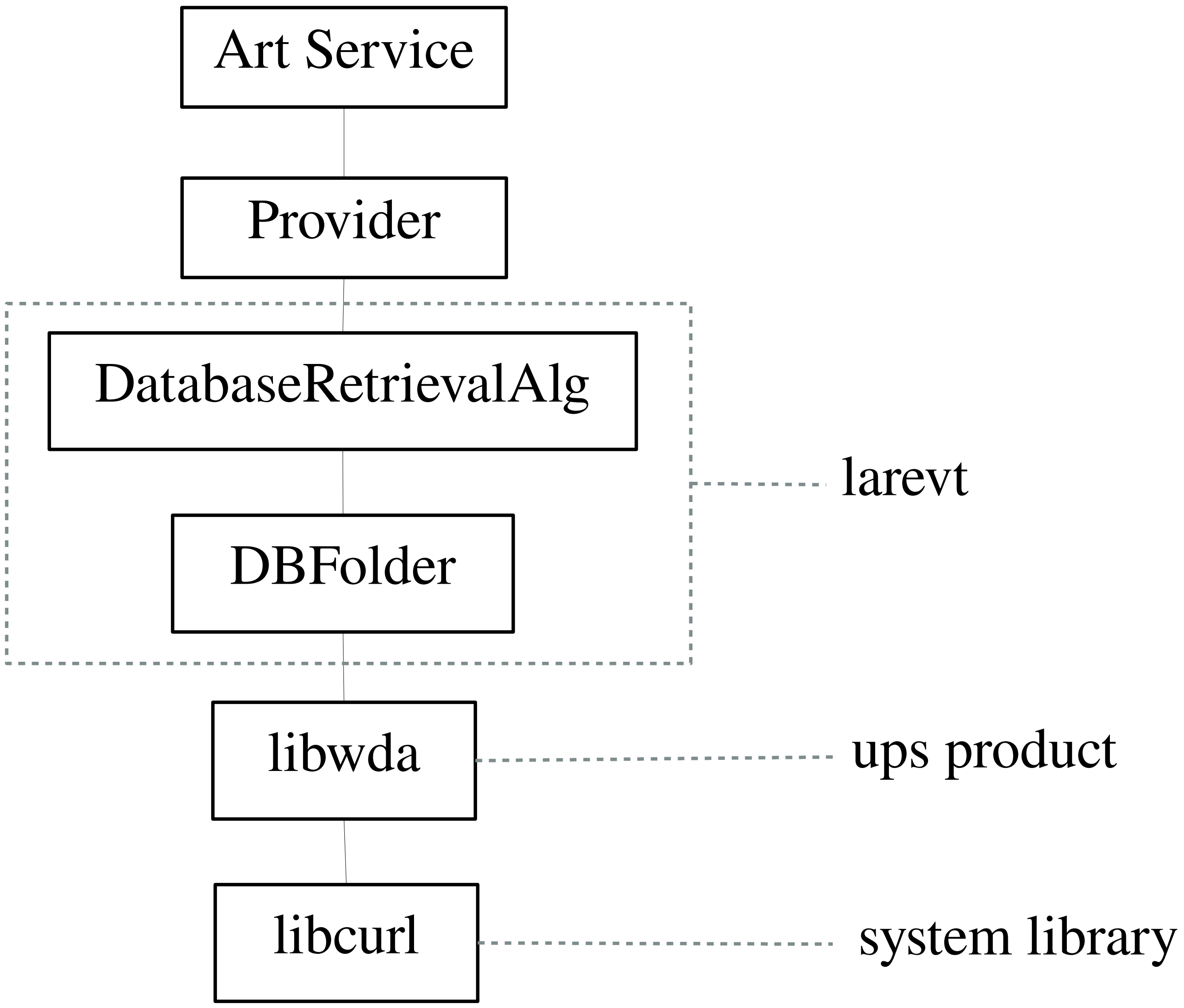
H. Greenlee

Contents

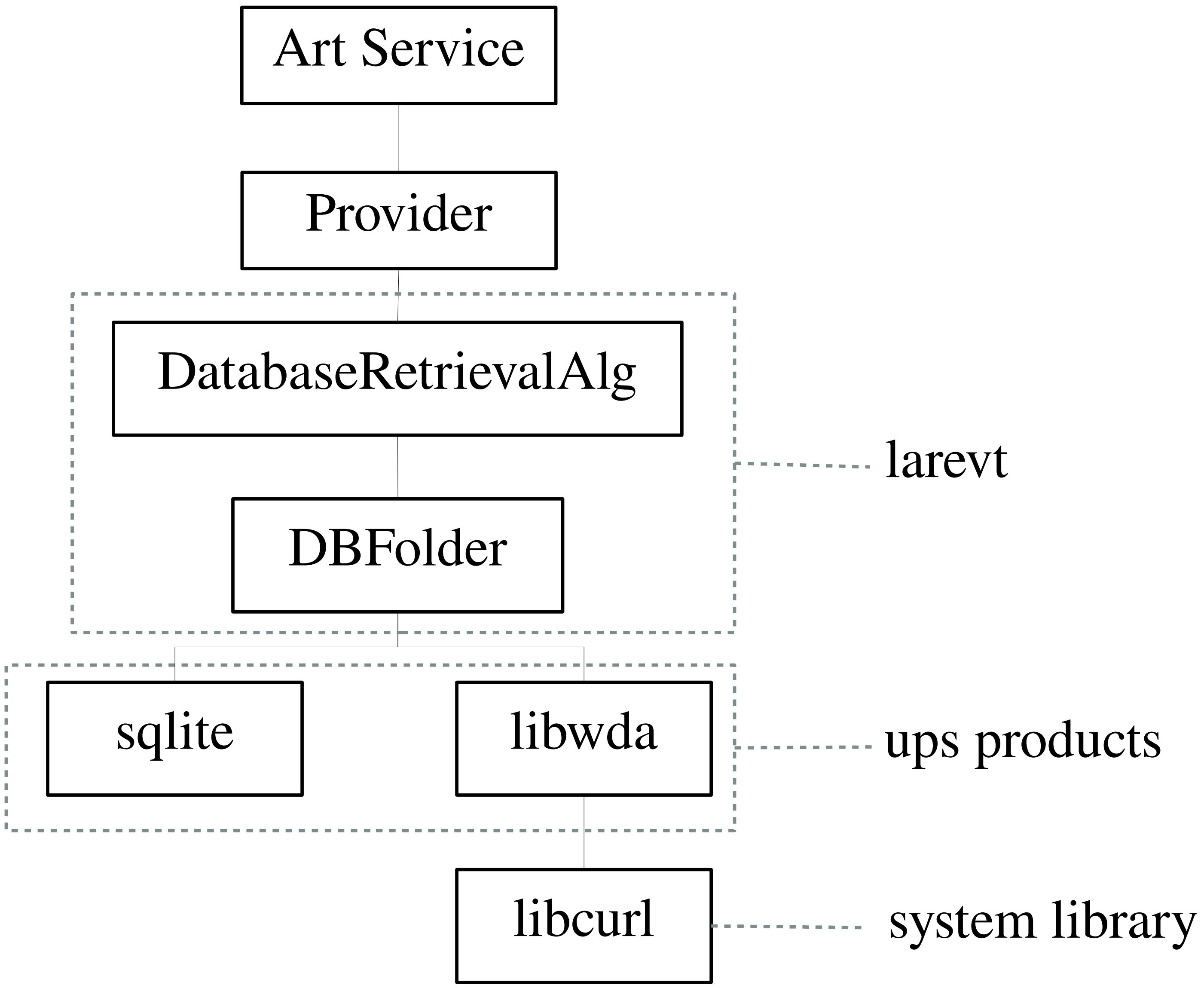
Overview of Calibration Database Access in Larsoft

- Calibration data are stored in a postgres interval-of-validity database.
  - Standard schemas exist for single-interval-of-validity (SIOV) and multiple-interval-of-validity (MIOV) databases.
  - MicroBooNE has 20 SIOV calibration databases.
- Larsoft includes an art service DatabaseUtil for direct access.
  - Little used.
- Most (all?) larsoft calibration database access makes use of http database servers.
- In this talk, I consider the option of exporting calibration data to an sqlite database (database-in-a-file).

Current Database Access Software Stack



Revised Database Access Software Stack



Historical Overview

- The last presentation that I made on this topic was in the Mar. 10, 2020 larsoft coordination meeting.
- At that meeting, I made two requests to the maintainers of libwda.
  - Add ability to search for server certificates in a directory.
    - This request was implemented in libwda v2_28_0.
  - Expose libwda opaque data struct (make header public).
    - This request was rejected (more about this issue later in this talk).
- Because of the opaque data struct issue, my proposed update to larevt at that meeting was never merged into larevt develop branch.
  - Nevertheless, MicroBooNE did update our larevt MCC9 branch, in which we cracked the libwda opaque data struct by copying the header into DBFolder.cxx.
  - MicroBooNE is using this version in production.

Previous Proposed Updates to Larevt

- Update DatabaseRetrievalAlg to add optional fcl parameter to choose between libwda and sqlite database access.
  - No update to individual providers or services is required, except fcl configuration updates.
- Update DBFolder to have ability to read from either libwda or sqlite.
  - This update is minimal in the sense that libwda access code is basically unchanged. Most changes are additions to read from sqlite.

New Proposed Updates to Larevt

- No additional proposed changes to DatabaseRetrievalAlg.
- Additional updates to DBFolder.
  - Change caching strategy to switch from storing data using the libwda opaque struct, to using a newly defined data structure.
  - This is a more invasive change than the old proposal in the sense that it affects both libwda and sqlite database access code.

How Libwda Works

- Libwda makes a database query by sending a specially formatted url to the database server.
- The server replies with a long text string that is a list of comma-separated values.
- Libwda chops the csv string into pieces and stores the resulting substrings in a an opaque c-struct (shown on following slide).
  - Struct header is hidden.
  - Libwda provides its own api for retrieving binary data from the opaque struct.

Libwda Opaque Struct

Start validity time			
End Validity Time			
Name 1	Name 2	Name 3	Name 4
Type 1	Type 2	Type 3	Type 4
Channel 1	Value	Value	Value
Channel 2	Value	Value	Value
Channel 3	Value	Value	Value

Libwda Opaque Struct

- The libwda struct is a dynamic two-dimensional array.
- All data is stored as strings.
  - String-to-binary conversion happens when data is extracted using the struct api.
- Libwda uses c-style memory management and c-style strings.

Libwda Struct Issues

- DBFolder uses the libwda opaque struct to cache database data.
  - This decision was made early before I got involved.
  - The original author of DBFolder found it necessary to partially crack the opaque struct.
- Adding sqlite capability made the opaque problem worse by making it necessary to update the cache with data read from sqlite.
  - Libwda does not supply an api to do this. The authors refused my request to add one.
- Opaqueness is not the only problem.
  - The current cache structure wastes memory by storing numeric values as strings, and by dynamically allocating memory for each value.

A Strategy for Solving All Libwda Struct Issues

- Instead of using the libwda struct for its database cache, DBFolder should use its own-defined data structure.
  - Removes need to go beyond current libwda api or to crack the libwda struct by copying the header (including the previously copied partial header).
  - Replacement data structure can be more memory-efficient.
    - Store numeric data in binary format.
    - Allocate memory for values at compile time.
  - A key design question is how to store arbitrary-type values.

Storing Arbitrary-Type Values in C++

- char*
  - This is what libwda does.
- std::string
- void*
  - Not type safe & no automatic destruction.
- boost::any
  - Type safe & automatic destruction.
- union
  - Memory allocated at compile time.
  - Not type safe & no automatic destruction.
- boost::variant<T1, T2,...>
  - Type safe, memory allocated at compile time, automatic destruction.
  - boost::variant<long, double, char[])
  - boost::variant<long, double, std::string)
  - boost::variant<long, double, unique_ptr<std::string>>

Memory Considerations

- char*, void*, unique_ptr<T>
  - 8 bytes + dynamic memory.
- std::string
  - 32 bytes + dynamic memory.
- union
  - Size of largest data member.
- boost::any
  - 16 bytes + dynamic memory.
- boost::variant<T1, T2, ...>
  - 8 bytes + size of largest template type.

New Database Cache Class DBDataset

- In the branch I am proposing, I added a new class DBDataset, which replaces the libwda struct as the database cache.

```
class DBDataset
{
public:

  typedef boost::variant<long, double, std::unique_ptr<std::string> > value_type;

  ...

private:

  IOVTimeStamp fBeginTime;                // IOV begin time.
  IOVTimeStamp fEndTime;                  // IOV end time.
  size_t fNRows;                          // Number of rows.
  size_t fNCols;                          // Number of columns.
  std::vector<std::string> fColNames;     // Column names.
  std::vector<std::string> fColTypes;     // Column types.
  std::vector<DBChannelID_t> fChannels;   // Channels.
  std::vector<value_type> fData;          // Calibration data.
};
```

DBFolder Interface

- The current version of DBFolder interface provides five typed accessors.

```
int GetNamedChannelData(DBChannelID_t channel, const std::string& name, bool& data);
int GetNamedChannelData(DBChannelID_t channel, const std::string& name, long& data);
int GetNamedChannelData(DBChannelID_t channel, const std::string& name, double& data);
int GetNamedChannelData(DBChannelID_t channel, const std::string& name, std::string& data);
int GetNamedChannelData(DBChannelID_t channel, const std::string& name, std::vector<double>& data);
```

- Boolean values are stored in DBDataset as integers (type long).
  - Sqlite (unlike postgres) doesn't have a boolean data type.
- The last accessor (type<double>, storing multiple values in one database column) makes no sense to me.
  - We don't use it to access MicroBooNE calibration data. I have no way to test it.
  - I removed this function in the merge branch.
  - I can't be sure that no other experiment code is using it, though.
  - This is the only significant interface change of DBFolder.

Implementation Status

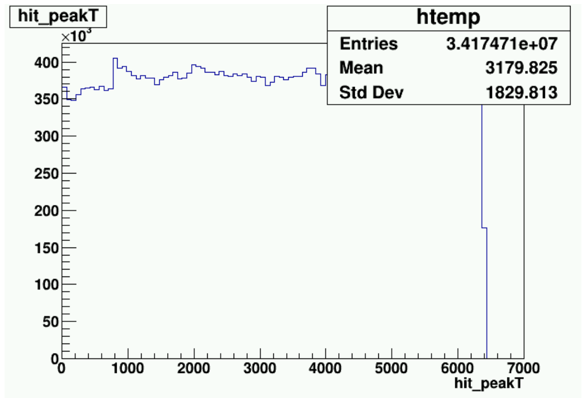
- The following merge branches are up to date with larsoft v09_08_00 and ready to merge to develop from uboone fork.
  - larevt: feature/greenlee_sqlite_develop
- There are also some uboone suite merge branches.
  - ubevt: feature/greenlee_sqlite_develop
  - uboonedata: feature/greenlee_sqlite_develop
  - uboonecode: feature/greenlee_sqlite_test_develop
- Other experiments don't need to make any updates, in which case they will continue to use libwda.
  - To use sqlite, they will need to add sqlite databases and make fcl updates.
- In uboonecode, we have added integration tests to ensure that sqlite databases get updated when new tags are added to postgres.

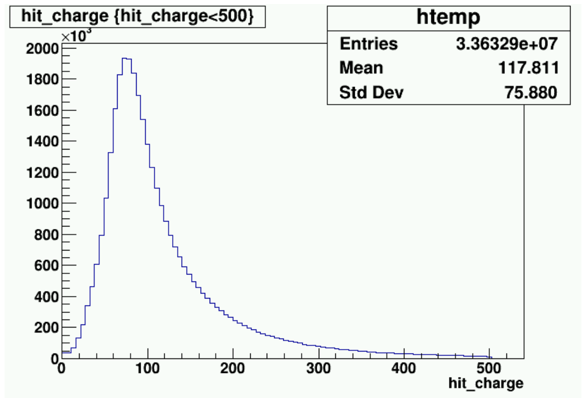
Validation Tests

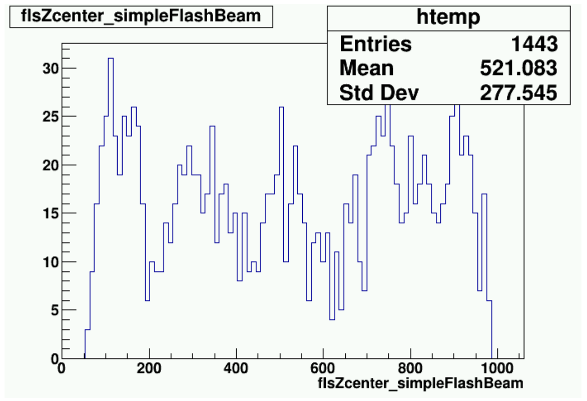
- Event dumps of calibration data.
- Running O(1000) events in test mode.
  - Read calibration data from libwda and sqlite and do binary comparision.
- Compare plots from vanilla v09_08_00 and updated larevt.
  - Ran reco1 and reco2 on same data events.

Comparison Plots
Hits & Flashes

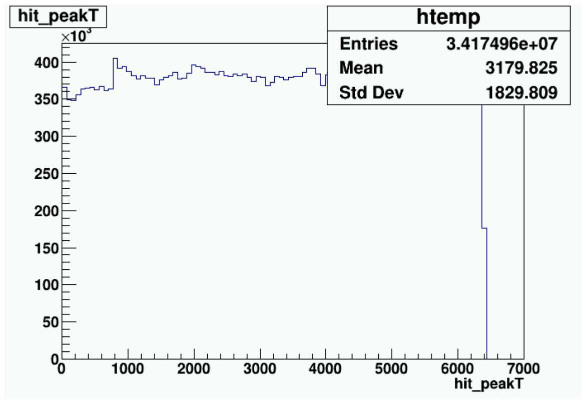
v09_08_00

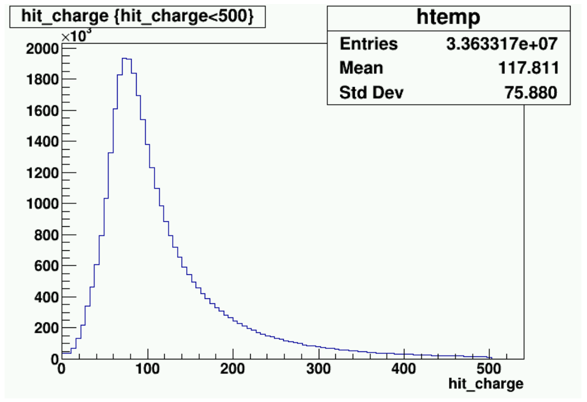


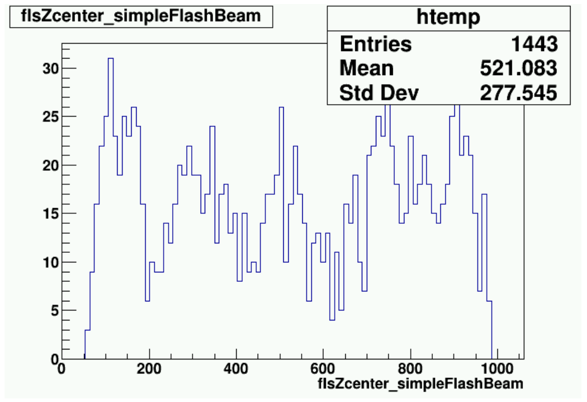




Patched

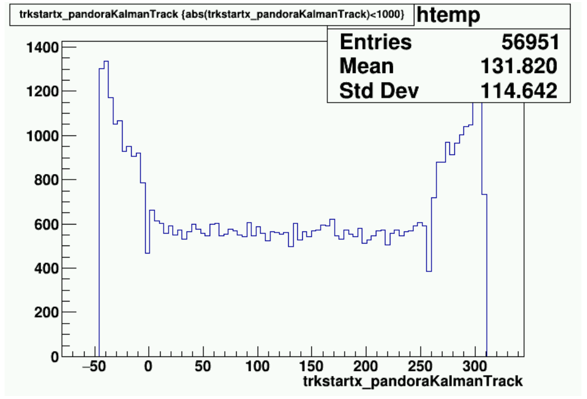


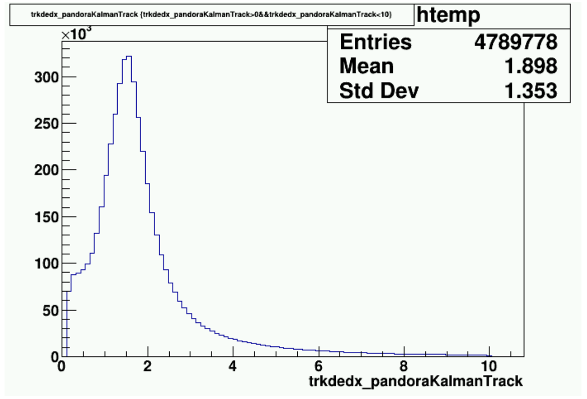




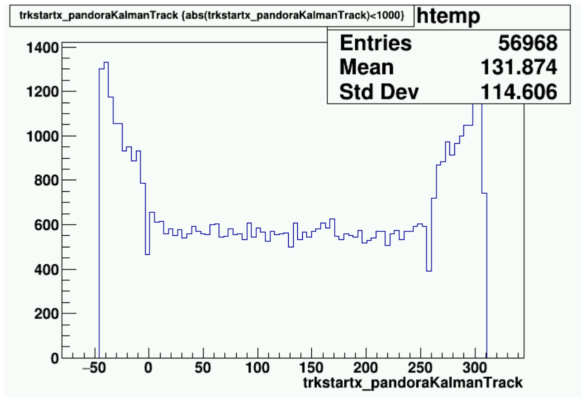
Comparison Plots
Tracks

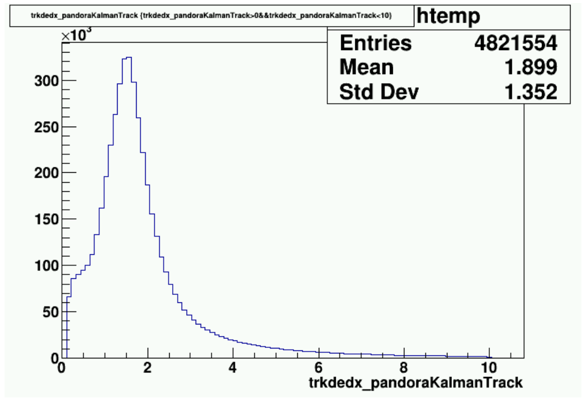
v09_08_00





Patched





Database Conversion Scripts

- For the record, MicroBooNE's postgres conversion scripts can be found in redmine repositiry ubutil/scripts (branch develop or v08_00_00_br).
  - siov_extracter.py
  - siov_extracter_sparsify.py

Summary

- Larevt branch greenlee_sqlite_develop from uboone github fork is ready to merge to larsoft develop branch.
  - I haven't made a pull request yet.
  - Caveat whether vector<double> DBFolder accessor is actually needed by experiments other than MicroBooNE.