Electronics Calibrations Database Interface

Brandon Eberly

December 6, 2016

Introduction

- Created larsoft provider+service interfaces for retrieving tpc electronics calibration information
  - Gain, shaping time, and extra experiment-specific info
  - Created default implementations in the MicroBooNE style that return gain, shaping time, and no extra info
  - **larevt** feature branch **eberly_asicDBI**

- Created MicroBooNE implementations
  - Return gain, shaping time, and extra info "is_misconfigured"
  - **uboonecode** feature branch **eberly_asicDBI**

- Modified DBFolder class to be able to retrieve booleans from a postgresql database (larevt: eberly_asicDBI)

- Modified CalibrationExtraInfo class to be able to hold booleans (larevt: eberly_asicDBI)

Provider Interface

- ElectronicsCalibProvider.h

```
/**
 \class ElectronicsCalibProvider
 * Currently, the class provides interface for the following information:
 * - electronics gain and its error
 * - electronics shaping time and its error
 * - electronics extra info, related to procedure that determines the gain and shaping time
 */
class ElectronicsCalibProvider {

  public:

    virtual ~ElectronicsCalibProvider() = default;

    /// Retrieve pmt gain information
    virtual float Gain(raw::ChannelID_t ch) const = 0;
    virtual float GainErr(raw::ChannelID_t ch) const = 0;
    virtual float ShapingTime(raw::ChannelID_t ch) const = 0;
    virtual float ShapingTimeErr(raw::ChannelID_t ch) const = 0;

    virtual CalibrationExtraInfo const& ExtraInfo(raw::ChannelID_t ch) const = 0;
};
```

Service Interface

- ElectronicsCalibService.h

```
/**
 \class ElectronicsCalibService
 This service provides only a simple interface to a provider class
 */
class ElectronicsCalibService {

  public:
    using provider_type = ElectronicsCalibProvider;

    /// Destructor
    virtual ~ElectronicsCalibService() = default;

    //retrieve provider
    ElectronicsCalibProvider const& GetProvider() const
    { return DoGetProvider(); }

    ElectronicsCalibProvider const* GetProviderPtr() const
    { return DoGetProviderPtr(); }

  private:

    /// Returns a reference to the service provider
    virtual ElectronicsCalibProvider const& DoGetProvider() const = 0;

    virtual ElectronicsCalibProvider const* DoGetProviderPtr() const = 0;



}; // class ElectronicsCalibService
```

DBFolder Modification

- Reminder: DBFolder is a class used in all MicroBooNE-style implementations of the provider+service interface classes

- Added a new overloaded GetNamedChannelData for bool types

```
int DBFolder::GetNamedChannelData(DBChannelID_t channel, const std::string& name, bool& data) {

  Tuple tup;
  size_t col = this->GetTupleColumn(channel, name, tup);
  int err=0;
  char buf[kBUFFER_SIZE];
  int str_size = getStringValue(tup, col, buf, kBUFFER_SIZE, &err);
  data = false;
  if (std::string(buf, str_size)=="True") {
    data = true;
  }
  else if (std::string(buf, str_size)=="False") {
    data = false;
  }
  else std::cout<<"(DBFolder) ERROR: Can't identify data: "<<std::string(buf, str_size)<<" as boolean!"<<std::endl;

  releaseTuple(tup);
  return err;
}
```

DBFolder Modification

- Added a check in the long GetNamedChannelData to handle case when user accidentally uses it to retrieve a bool:

```
int DBFolder::GetNamedChannelData(DBChannelID_t channel, const std::string& name, long& data) {

  Tuple tup;
  size_t col = this->GetTupleColumn(channel, name, tup);
  int err=0;

  //first handle special case that the db data is boolean, but user mistakenly used long version of this function
  char buf[kBUFFER_SIZE];
  int str_size = getStringValue(tup, col, buf, kBUFFER_SIZE, &err);
  if (std::string(buf, str_size)=="True") {
    data = 1;
  }
  else if (std::string(buf, str_size)=="False") {
    data = 0;
  }
  else { //ok, we really have a long (hopefully)
    data = getLongValue(tup, col, &err);
  }
  releaseTuple(tup);
  return err;
}
```

CalibrationExtraInfo Modification

- A relatively new class: containers a series of containers for ints, floats, strings, etc. that are keyed by strings.
  - Meant to hold experiment-specific calibration data in a way that can be returned by experiment-agnostic interfaces

- Modification: store and return bools

```
void CalibrationExtraInfo::AddOrReplaceBoolData(std::string const& label, bool const data) {
  fBoolData[label] = data;
}
```

```
bool CalibrationExtraInfo::GetBoolData(std::string const& label) const {
  if (fBoolData.find(label) != fBoolData.end()) {
    return fBoolData.at(label);
  }

  throw IOVDataError("CalibrationExtraInfo: Could not find extra bool data "+label+" for calibration "+fName);

}
```

- Also changed functions for clearing the member containers