# Data Types: Serialization and Versioning, courtesy of Boost

7,14(pg6)-Nov-2012, Eric Church, Yale

# Boost library

- Need to accommodate being able to read back old data structures, while creating new ones: adding features to output classes.
- Boost serializing/versioning takes care of this in one fell swoop

# Imagine we decide we want to add the string driver_name

```
#include <boost/serialization/list.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/version.hpp>
#include <boost/serialization/split_member.hpp>

class gps_position
{
    friend class boost::serialization::access;
    int degrees;
    int minutes;
    int seconds;
    int milliseconds;
    int microseconds;

    std::string driver_name;

    template<class Archive>
    void save(Archive & ar, const unsigned int version) const
    {
        // note, version is always the latest when saving
        ar  & driver_name;
        ar  & degrees;
        ar  & minutes & seconds & milliseconds & microseconds;
    }
    template<class Archive>
    void load(Archive & ar, const unsigned int version)
    {
        if(version > 0)
            ar & driver_name;
        ar  & degrees;
        ar  & minutes & seconds & milliseconds & microseconds;
    }
    BOOST_SERIALIZATION_SPLIT_MEMBER()
 public:
    gps_position(){};
    gps_position(int d, int m, float s) :
        degrees(d), minutes(m), seconds(s)
    {}

};

BOOST_CLASS_VERSION(gps_position, 1)
```

The macro `BOOST_SERIALIZATION_SPLIT_MEMBER()` generates code which invokes the `save` or `load` depending on whether the archive is used for saving or loading.

class Version number

```
int main() {
    // create and open a character archive for output
    std::ofstream ofs("filename");

    // create class instance
    const gps_position g(35, 59, 24.567f);

    // save data to archive
    {
        std::ofstream ofs("filename");
        boost::archive::text_oarchive oa(ofs);
        // write class instance to archive
        oa & g; // save, equivalent to oa << g
       // archive and stream closed when destructors are called
    }

// ... some time later restore the class instance to its original state
gps_position newg;

// load data from archive
{
    // create and open an archive for input
    std::ifstream ifs("filename");
    boost::archive::text_iarchive ia(ifs);
    // read class state from archive
    ia & newg; // load, equivalent to ia >> newg
    // archive and stream closed when destructors are called
}
return 0;
```

# Conclusion

- Is there any reason not to use the boost libraries?
- If not, after iterating once more with Cat, I will start coding the data classes

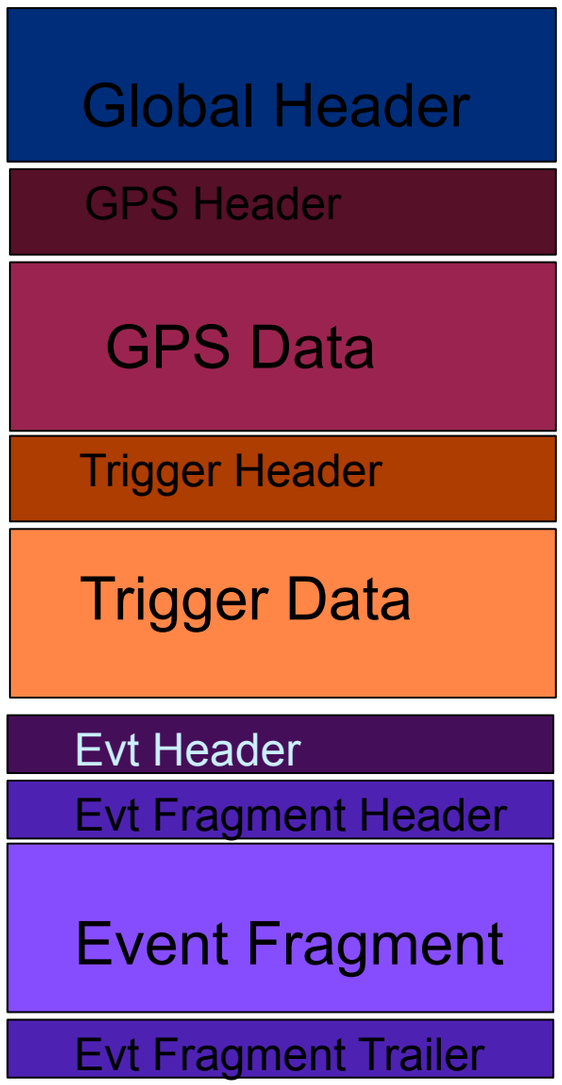
# Actual Work, as pertains to the uB DAQ

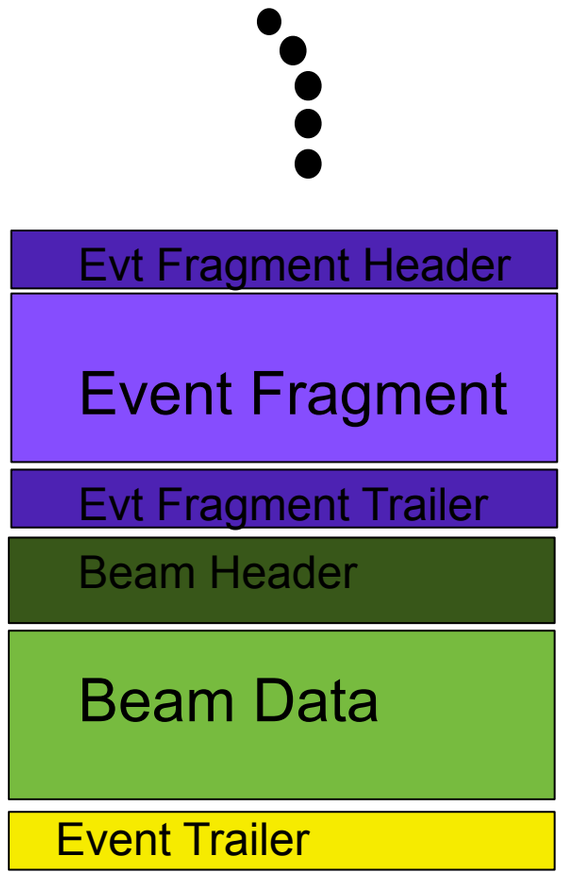
- 14-Nov-2012

# C++ Structs: Headers, data.

**subBlockHeaders:**

Trigger Header,
GPS Header,
Evt Header,
Beam Header




There are Ncrates Fragments.

# subBlockHeader

```
class subBlockHeader
{
private:
  friend class boost::serialization::access;


  template<class Archive>
    void save(Archive & ar, const unsigned int version) const
    {
      if (version == 0 )
        ar  & sizeDataFollowing & subBlockType;
      else if (version == 1)
        ar  & sizeDataFollowing & subBlockType & NHL_lockout;
    }

  template<class Archive>
    void load(Archive & ar, const unsigned int version)
    {
      if(version == 0) // This will change for reading old files.
        ar & sizeDataFollowing & subBlockType;
      else if (version == 1)
        ar & sizeDataFollowing & subBlockType & NHL_lockout;
    }
  BOOST_SERIALIZATION_SPLIT_MEMBER()

public:
  subBlockHeader(){};
  subBlockHeader(int _size, std::string _type, bool _nhl):
    sizeDataFollowing(_size), subBlockType(_type), NHL_lockout(_nhl)
    {};

  int sizeDataFollowing;
  std::string subBlockType;
  bool NHL_lockout;
};
```

# tests

```
int main() {
    // create and open a character archive for output
    std::ofstream ofs("filename3b");
    //    boost::archive::text_oarchive oa(ofs);
    boost::archive::binary_oarchive oa(ofs);

    // save data to archive
    std::vector<int> numbers = {1,4,5,12};
    for (auto ii : numbers)
    {
      // create class instance, writing latest version data,
      const datatypes::subBlockHeader g(12+ ii,"GPS",true);

      // write class instance to archive
      oa & g; // save, equivalent to oa << g
      // archive and stream closed when destructors are called
    }
    ofs.close();

    // ... some time later restore the class instance to its original state

    // load data from archive
    std::ifstream ifs("filenameb");
    //    boost::archive::text_iarchive ia(ifs);
    boost::archive::binary_iarchive ia(ifs);
    for (auto ii : numbers)
    {

      datatypes::subBlockHeader newg;
      // read class state from archive
      ia & newg; // load, equivalent to ia >> newg
      // archive and stream closed when destructors are called
      std::cout << "write_read recovers subBlockType " << ii << " as " << newg.subBlockType << std::endl;
      std::cout << "write_read recovers sizeDataFollowing as " << newg.sizeDataFollowing << std::endl;
      std::cout << "write_read recovers lockout as " << newg.NHL_lockout << std::endl;
    }
    ifs.close();
```

instantiate, fill class & write out binary data of one version

instantiate, read old data into same class, different version

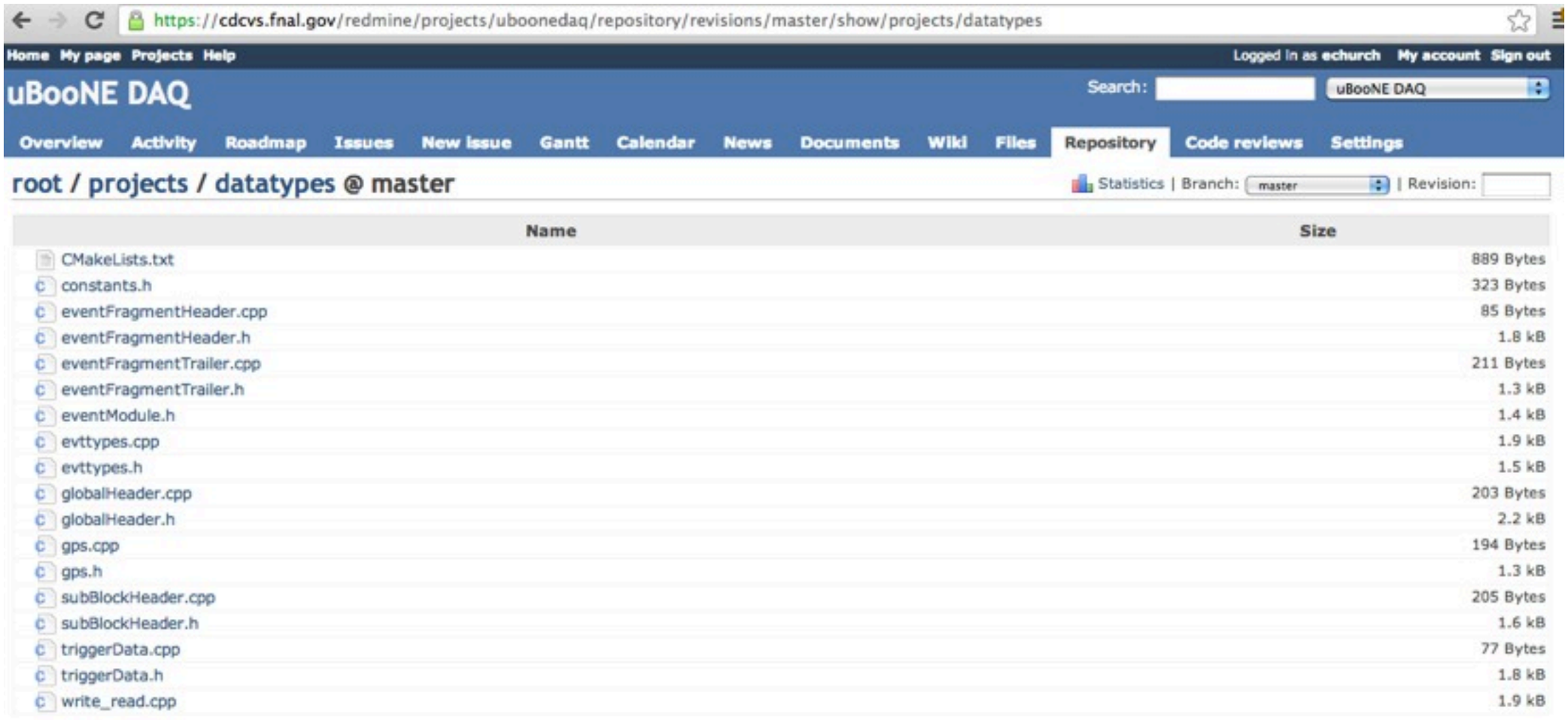
# datatypes/EventFragmentHeader.cpp,h

const int VERSION=0;

version serialization managed by boost

```
#ifndef _UBOONETYPES_EVENTFRAGMENTHEADER_H
#define _UBOONETYPES_EVENTFRAGMENTHEADER_H
#include <sys/types.h>
#include <inttypes.h>
#include "evttypes.h"
#include "gps.h"

#include <boost/serialization/list.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/version.hpp>

#include "constants.h"

namespace gov {
namespace fnal {
namespace uboone {
namespace datatypes {

using namespace gov::fnal::uboone;


class eventFragmentHeader {
 private:
  friend class boost::serialization::access;


  template<class Archive>
    void save(Archive & ar, const unsigned int version) const
    {
      ar & size & crate_number & card_address & event_number & frame_number & gps10 ;
    }

  template<class Archive>
    void load(Archive & ar, const unsigned int version)
    {
      if(version == 0) // This will change for reading old files.
        ar & size & crate_number & card_address & event_number & frame_number & gps10 ;
      else if (version == 1)
        ar & size & crate_number & card_address & event_number & frame_number & gps10 ;
    }
  BOOST_SERIALIZATION_SPLIT_MEMBER()

    public:
  static const uint8_t DAQ_version_number = gov::fnal::uboone::datatypes::constants::VERSION;

  uint32_t size; //bytes, needs to be uint32_t for large events
  uint16_t crate_number; // Crate #
  uint16_t card_address; // Card count
  uint16_t event_number; // Event #
  uint16_t frame_number; // Frame #
  gps    gps10; // from GPS card! At least in crate 10.

  eventFragmentHeader();
};
BOOST_CLASS_VERSION(gov::fnal::uboone::datatypes::eventFragmentHeader,
gov::fnal::uboone::datatypes::constants::VERSION)

```

# projects/datatypes



# To Do

- Make some changes to finalize 0th order class definitions in datatypes
- Get a draft Beam Data class in. From Zarko.
- Cut over in seb and assembler code to using these classes