

# 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

1

```

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;
}

```

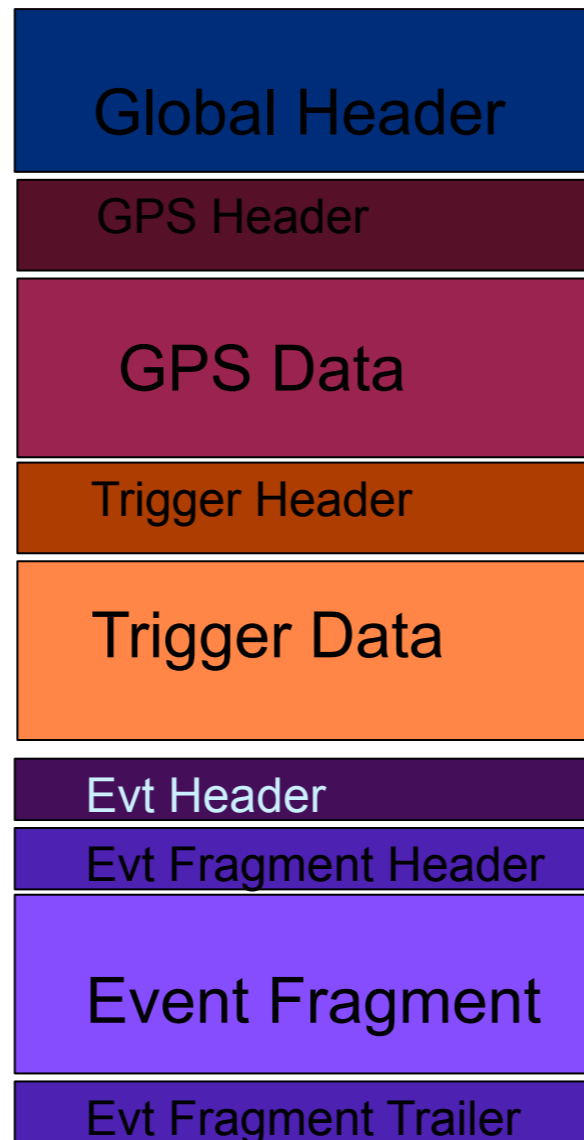
# Conclusion

- 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

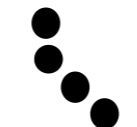
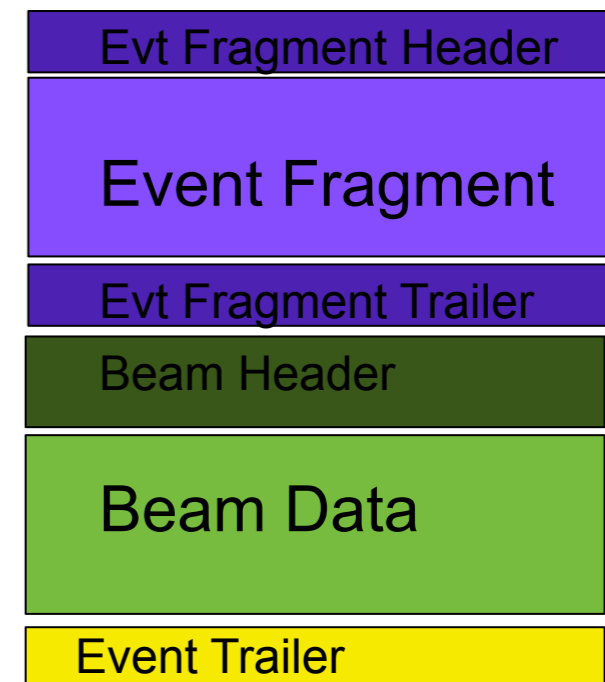
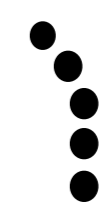
- 14-Nov-2012

# 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: versions

```
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

# tests: speed

```
struct subt {
    uint32_t size;
    std::string type;
    bool demo;
    subt(uint32_t _size, std::string _type, bool _demo)
    {
        size=_size; type=_type; demo=_demo;
    };
};
```

```
std::ofstream ofbsv("filenamebv");
std::ofstream ofbs("filenameb");
```

```
boost::archive::binary_oarchive oab(ofbs);
const datatypes::subBlockHeader g(12,"GPS",true);
const subt gs(12,"XYZ",true);
int64_t sz(0);
nLoops=0;
```

```
while (nLoops<maxLoops)
{
    oab & g; // save, equivalent to oa << g
    sz+=sizeof(g);
    nLoops++;
}
```

```
clock_gettime(CLOCK_REALTIME, &ts1);
ulfinishNs = ts1.tv_sec * 1.0E9 + ts1.tv_nsec;
```

```
int64_t sz2(0);
nLoops=0;
```

```
while (nLoops<maxLoops)
{
    ofbsv << gs.size << ", " << gs.type << ", " << gs.demo << ". ";
    sz2+=sizeof(gs);
    nLoops++;
}
```

```
clock_gettime(CLOCK_REALTIME, &ts2);
ulfinishNs2 = ts2.tv_sec * 1.0E9 + ts2.tv_nsec;
```

# speed2

```
[echurch@ubooneDAQ-seb-01 ubooneDAQ]$ write_read  
sizeof(g) 24  
sizeof(sizeDataFollowing) 4  
sizeof(subBlockType) 8  
sizeof(NHL_lockout) 1  
write_read boost::archive::serial write reads 0.0148029 GBytes/sec  
time is 1.6213 sec
```

```
sizeof(gs) 24  
sizeof(size) 4  
sizeof(type) 8  
sizeof(demo) 1  
write_read Vanilla write reads 0.0117989 GBytes/sec  
time is 2.03409 sec
```


# speed 3: conclusion

So, not exactly apples  
to apples, but close.  
No obvious slow  
down to write to  
boost::archive::serial  
files.

# datatypes/EventFragmentHeader.cpp,h

```
1 #ifndef _UBOONETYPES_EVENTFRAGMENTHEADER_H
2 #define _UBOONETYPES_EVENTFRAGMENTHEADER_H
3 #include <sys/types.h>
4 #include <inttypes.h>
5 #include "evttypes.h"
6 #include "gps.h"
7
8 #include <boost/serialization/list.hpp>
9 #include <boost/serialization/string.hpp>
10 #include <boost/serialization/version.hpp>
11
12 #include "constants.h" ← const int VERSION=0;
13
14 namespace gov {
15     namespace fnal {
16         namespace uboone {
17             namespace datatypes {
18
19                 using namespace gov::fnal::uboone;
20
21
22                 class eventFragmentHeader {
23                 private:
24                     friend class boost::serialization::access;
25
26
27                     template<class Archive>
28                     void save(Archive & ar, const unsigned int version) const
29                     {
30                         ar & size & crate_number & card_address & event_number & frame_number & gps10 ;
31                     }
32
33                     template<class Archive>
34                     void load(Archive & ar, const unsigned int version)
35                     {
36                         if(version == 0) // This will change for reading old files.
37                             ar & size & crate_number & card_address & event_number & frame_number & gps10 ;
38                         else if (version == 1)
39                             ar & size & crate_number & card_address & event_number & frame_number & gps10 ;
40                     }
41                     BOOST_SERIALIZATION_SPLIT_MEMBER()
42
43                 public:
44                     static const uint8_t DAQ_version_number = gov::fnal::uboone::datatypes::constants::VERSION;
45
46                     uint32_t size; //bytes, needs to be uint32_t for large events
47                     uint16_t crate_number; // Crate #
48                     uint16_t card_address; // Card count
49                     uint16_t event_number; // Event #
50                     uint16_t frame_number; // Frame #
51                     gps      gps10; // from GPS card! At least in crate 10.
52
53                     eventFragmentHeader();
54
55                     BOOST_CLASS_VERSION(gov::fnal::uboone::datatypes::eventFragmentHeader,
56                     gov::fnal::uboone::datatypes::constants::VERSION)
57
58
59
60
61
```

version serialization  
managed by boost



# projects/datatypes

Home My page Projects Help Logged In as echurch My account Sign out

uBooNE DAQ Search:  uBooNE DAQ

Overview Activity Roadmap Issues New Issue Gantt Calendar News Documents Wiki Files **Repository** Code reviews Settings

root / projects / datatypes @ master Statistics | Branch: master | Revision:

Name	Size
CMakeLists.txt	889 Bytes
constants.h	323 Bytes
eventFragmentHeader.cpp	85 Bytes
eventFragmentHeader.h	1.8 kB
eventFragmentTrailer.cpp	211 Bytes
eventFragmentTrailer.h	1.3 kB
eventModule.h	1.4 kB
evttypes.cpp	1.9 kB
evttypes.h	1.5 kB
globalHeader.cpp	203 Bytes
globalHeader.h	2.2 kB
gps.cpp	194 Bytes
gps.h	1.3 kB
subBlockHeader.cpp	205 Bytes
subBlockHeader.h	1.6 kB
triggerData.cpp	77 Bytes
triggerData.h	1.8 kB
write_read.cpp	1.9 kB

# 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