A template for INTT calibration class

Takashi Hachiya
Nara Women’s University

Template for calibration module

- Purpose of Calibration Parameters (for INTT production)
  - Hot&DeadMap, BCO timing, and DAC
    - Masking bad hits using Hot＆Dead map
    - Masking bad hits using BCOFull - BCO
    - Converting ADC to DAC
      - Clustering should be performed w/ DAC weight (not ADC)
- Proposal
  - One class for one calibration parameter
  - All these classes will read the parameters from CDBTTree.
    - LoadFromCDB, LoadFromFile
  - Hot&Dead and BCO class needs **predicate (isBad)**
  - DAC class needs **accessor (GetDAC)**

  - “common function name” as implementation
    - I originally thought to use “class inheritance” to implement common functions, but decided not use it because of the disadvantages.

Common function names

- Accessing to/from CDBTTree
  - LoadFromFile(string const& calibname);
  - LoadFromCDB(string const& calibname);
  - WriteToFile(string const& calibname);

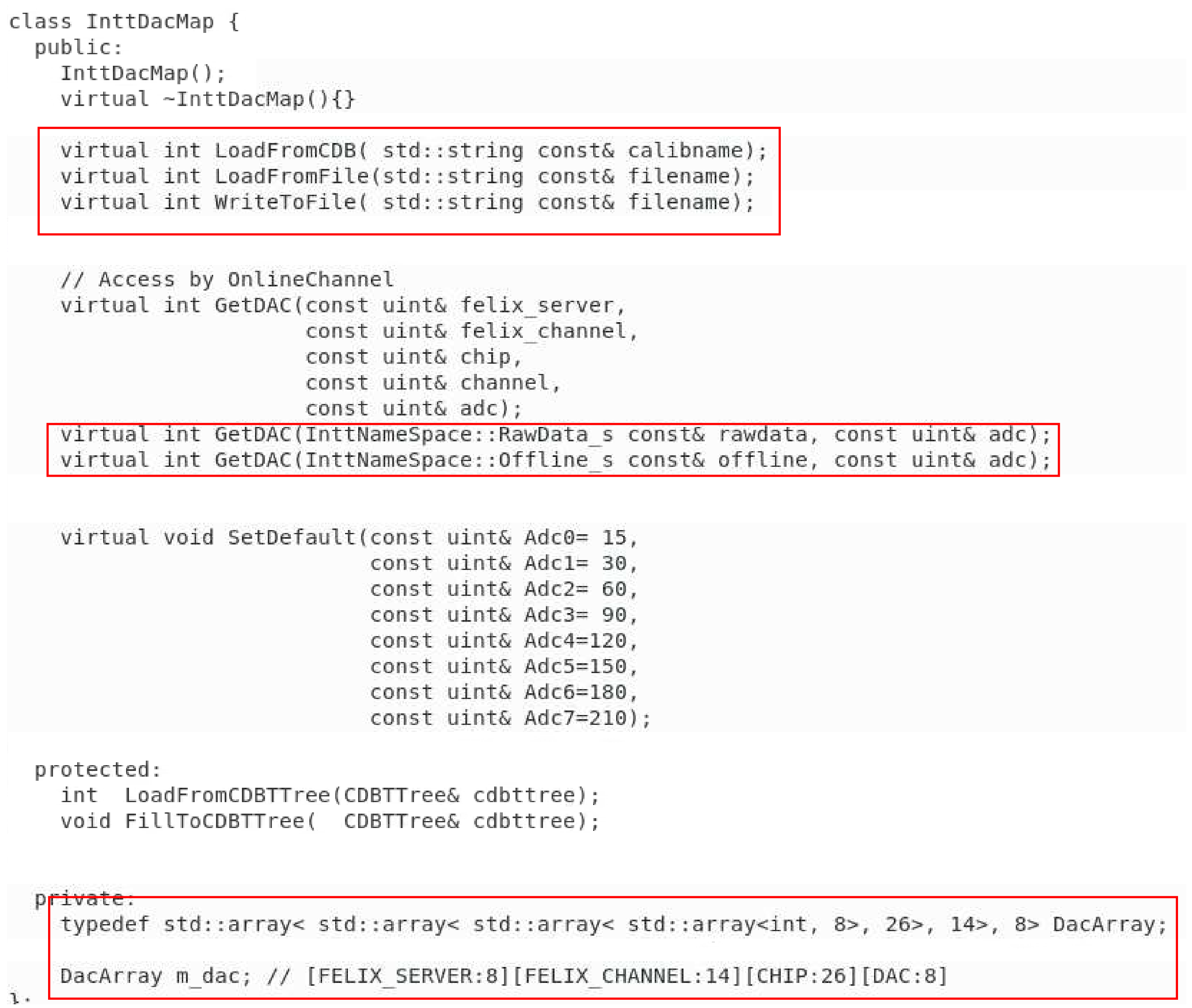
- Channel I/F should be both online and offline
  - Online ch. for data, offline ch. for simulation

  - IsBad for HotDead & BCO
    - bool isBad(RawData_s& rawdata)
    - bool isBad(Offline_s& offline)

  - GetDAC for DAC
    - int GetDAC(RawData_s& offline, int &adc)
    - int GetDAC(Offline_s& offline, int &adc)

- Leave implementation to those who develop

InttDacMap



```
class InttDacMap {
  public:
    InttDacMap();
    virtual ~InttDacMap(){}

    virtual int LoadFromCDB( std::string const& calibname);
    virtual int LoadFromFile(std::string const& filename);
    virtual int WriteToFile( std::string const& filename);

    // Access by OnlineChannel
    virtual int GetDAC(const uint& felix_server,
                       const uint& felix_channel,
                       const uint& chip,
                       const uint& channel,
                       const uint& adc);
    virtual int GetDAC(InttNameSpace::RawData_s const& rawdata, const uint& adc);
    virtual int GetDAC(InttNameSpace::Offline_s const& offline, const uint& adc);

    virtual void SetDefault(const uint& Adc0= 15,
                            const uint& Adc1= 30,
                            const uint& Adc2= 60,
                            const uint& Adc3= 90,
                            const uint& Adc4=120,
                            const uint& Adc5=150,
                            const uint& Adc6=180,
                            const uint& Adc7=210);

  protected:
    int  LoadFromCDBTTree(CDBTTree& cdbttree);
    void FillToCDBTTree(  CDBTTree& cdbttree);

  private:
    typedef std::array< std::array< std::array< std::array<int, 8>, 26>, 14>, 8> DacArray;

    DacArray m_dac; // [FELIX_SERVER:8][FELIX_CHANNEL:14][CHIP:26][DAC:8]
};
```

- DacArray is a fixed length array (stl::array)

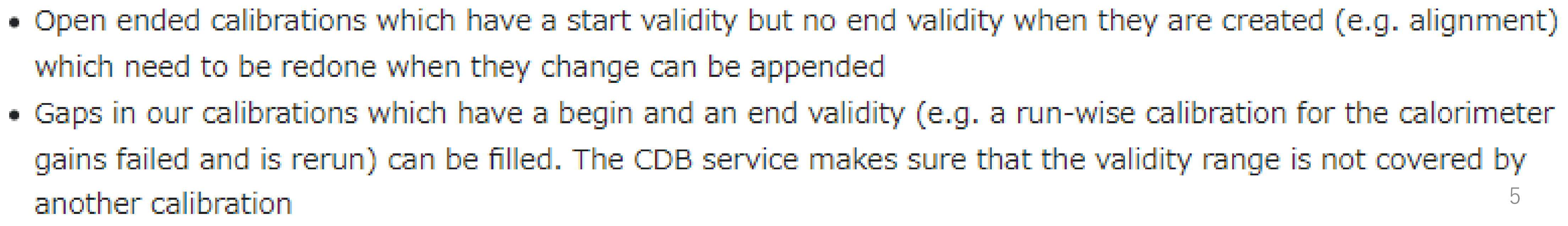
How to use in the unpacker

- Have the object in the unpacker
- Unpacker has a "set" function to know the calibration name
  - SetCalibDAC(string calibname, int flag); // flag = CDB or FILE
  - Constant Object in the macro is also OK

- The calibration object will be initialized in Unpacker::InitRun
  - RunNumber (time) is accessible in Unpacker::InitRun

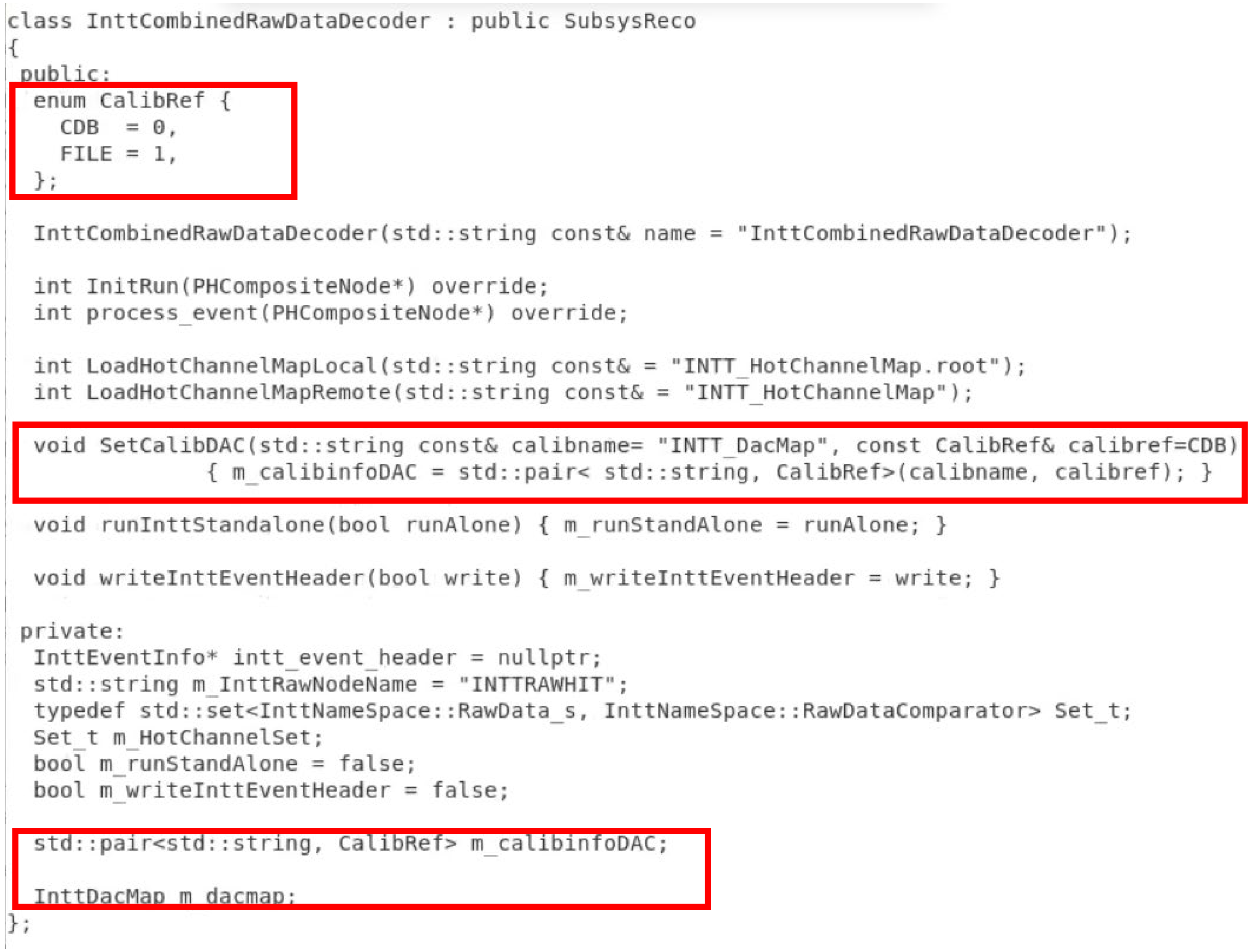
  - Calibration parameter will be identified by time

https://wiki.sphenix.bnl.gov/index.php/Calibrations-db



- Open ended calibrations which have a start validity but no end validity when they are created (e.g. alignment) which need to be redone when they change can be appended
- Gaps in our calibrations which have a begin and an end validity (e.g. a run-wise calibration for the calorimeter gains failed and is rerun) can be filled. The CDB service makes sure that the validity range is not covered by another calibration

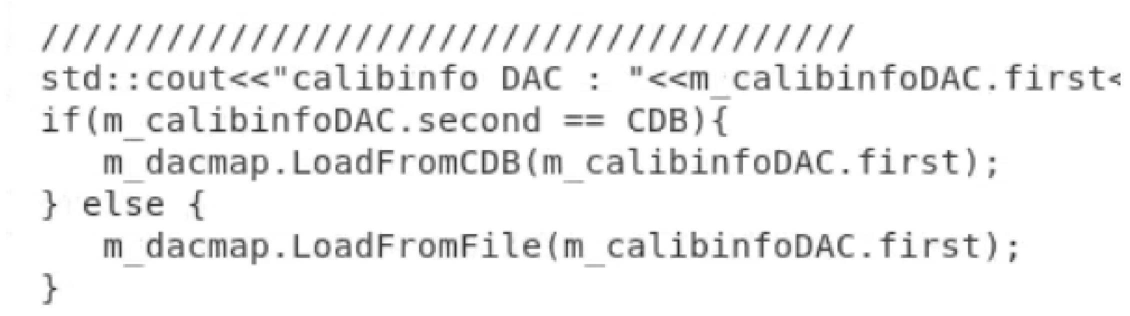
Unpacker: Implementation



```
class InttCombinedRawDataDecoder : public SubsysReco
{
 public:
  enum CalibRef {
    CDB  = 0,
    FILE = 1,
  };

  InttCombinedRawDataDecoder(std::string const& name = "InttCombinedRawDataDecoder");

  int InitRun(PHCompositeNode*) override;
  int process_event(PHCompositeNode*) override;

  int LoadHotChannelMapLocal(std::string const& = "INTT_HotChannelMap.root");
  int LoadHotChannelMapRemote(std::string const& = "INTT_HotChannelMap");

  void SetCalibDAC(std::string const& calibname= "INTT_DacMap", const CalibRef& calibref=CDB)
             { m_calibinfoDAC = std::pair< std::string, CalibRef>(calibname, calibref); }

  void runInttStandalone(bool runAlone) { m_runStandAlone = runAlone; }

  void writeInttEventHeader(bool write) { m_writeInttEventHeader = write; }

 private:
  InttEventInfo* intt_event_header = nullptr;
  std::string m_InttRawNodeName = "INTTRAWHIT";
  typedef std::set<InttNameSpace::RawData_s, InttNameSpace::RawDataComparator> Set_t;
  Set_t m_HotChannelSet;
  bool m_runStandAlone = false;
  bool m_writeInttEventHeader = false;

  std::pair<std::string, CalibRef> m_calibinfoDAC;

  InttDacMap m_dacmap;
};
```

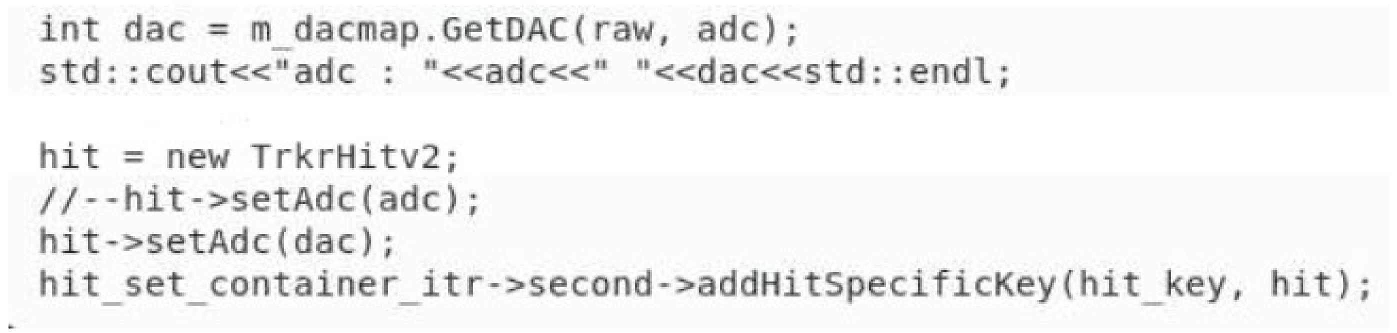
Unpacker: Implementation

InttCombinedRawdataDecoder::InitRun



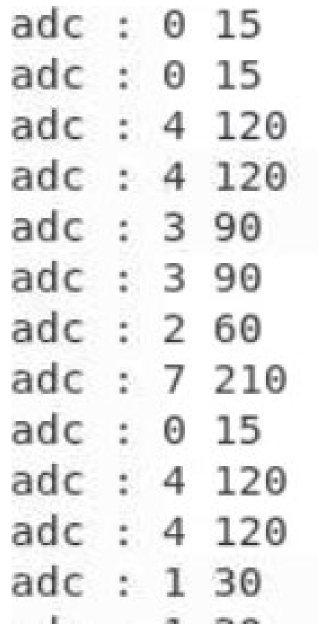
```
//////////////////////////////////////////
std::cout<<"calibinfo DAC : "<<m_calibinfoDAC.first<
if(m_calibinfoDAC.second == CDB){
   m_dacmap.LoadFromCDB(m_calibinfoDAC.first);
} else {
   m_dacmap.LoadFromFile(m_calibinfoDAC.first);
}
```

InttCombinedRawdataDecoder::process_event



```
int dac = m_dacmap.GetDAC(raw, adc);
std::cout<<"adc : "<<adc<<" "<<dac<<std::endl;

hit = new TrkrHitv2;
//--hit->setAdc(adc);
hit->setAdc(dac);
hit_set_container_itr->second->addHitSpecificKey(hit_key, hit);
```

Result



```
adc : 0 15
adc : 0 15
adc : 4 120
adc : 4 120
adc : 3 90
adc : 3 90
adc : 2 60
adc : 7 210
adc : 0 15
adc : 4 120
adc : 4 120
adc : 1 30
```