File Records Implemention

- **Motivation**
- **File records data store**
- **Place, retrieve and match objects**
- **Conclusions**

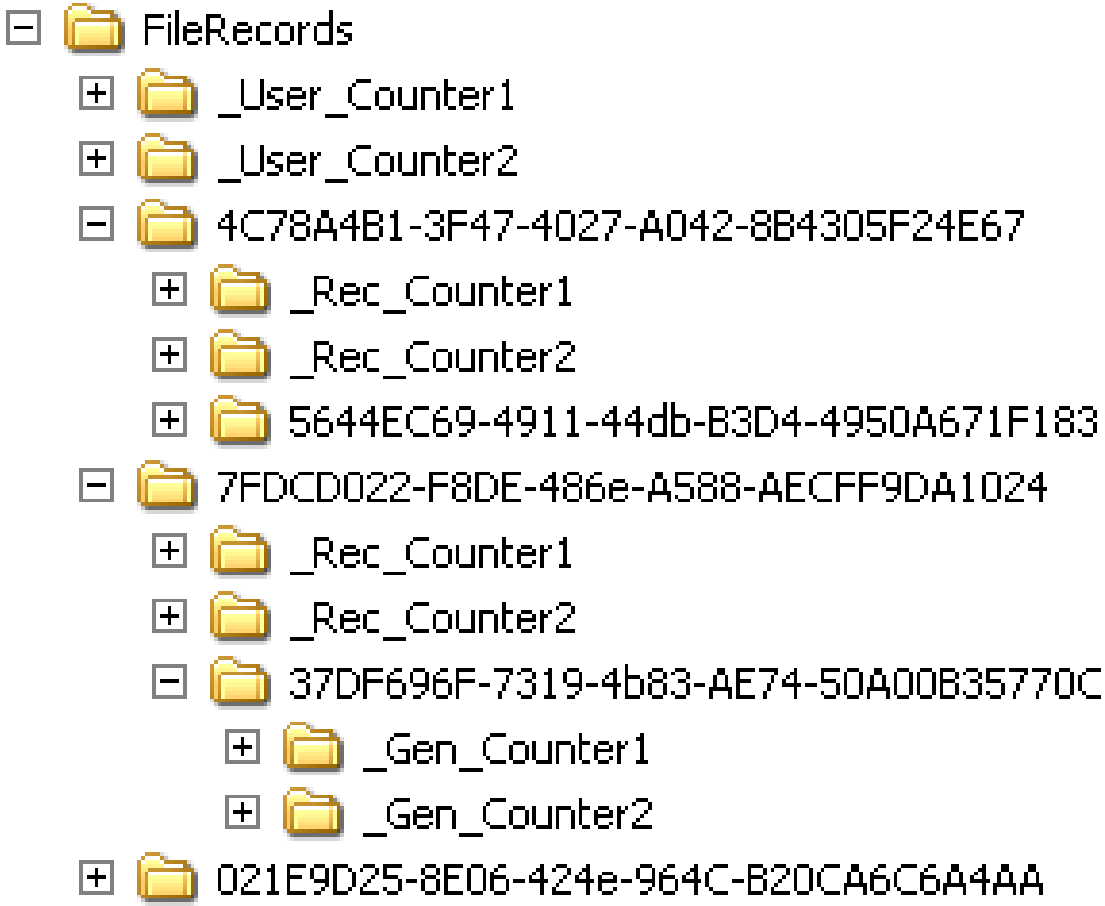
Motivation

- **Need to store data on a per file bases together with event data**
  - Typically counters, summary objects, etc.
  - But only ONCE, not for each event
- **Only user until now**
  Saving of luminosity estimates/measurements
  - Numbers need to survive stripping phase
  - Luminosity events are stripped off
- **Other use cases until today**
  - …partially contradicting and loosely specified except: "I want to save…"

Expected Behaviour

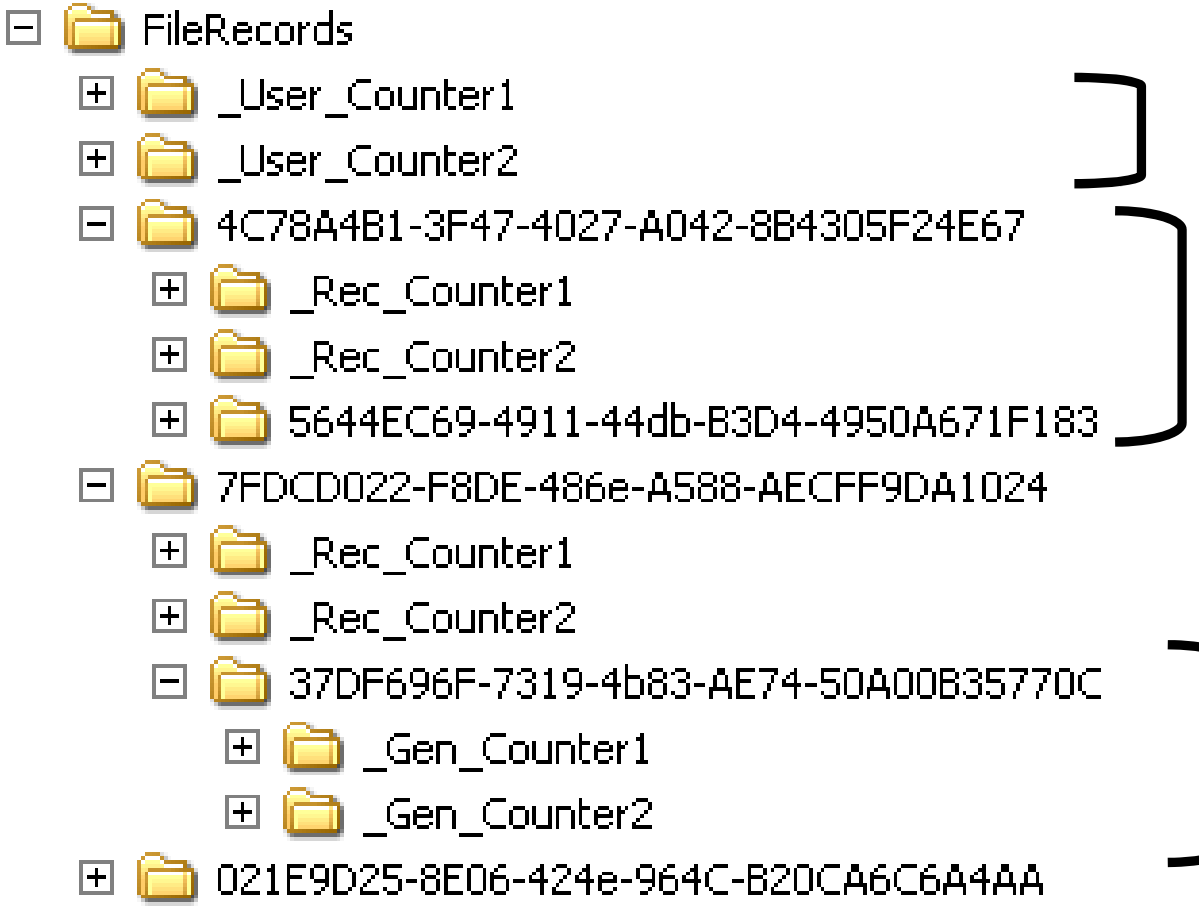
- **Object access like any other data service:**
  - Handle any DataObject
  - Known access mechanism to place and retrieve objects
- **To be saved with event data in the same file**
- **Allow to recursively propagate history counters from input files**
  - Mini-DST may contain records from
    - Brunel, Brunel includes records from Boole, Boole from Gauss...
- **Important notice**
  - Only works for files with direct object access (POOL)

FileRecord Store Layout



- **Tree structure like any DataSvc:**
  - Event data
  - Detector data
  - N-tuples,…
- **Contains DataObject(s) or subclasses**

FileRecord Store Layout



- **Root entry**
- **Current record entries**
- **History record from input file**
  - name is FID
  - now: input file
  - entries + history
- **Another input file**
- **Input record of inner input file**
  - entries + history
- **and another input file**

How to add Entities

- **Like other data store items**
- **Need to retrieve reference to file records service**
  - **sc=service("FileRecordDataSvc",m_recordsSvc)**
- **Then place object in TES**
  - Before services get finalized
  - Use either Algorithm **or** use “stop” transition!
- **GaudiAlg: get<T>(…), put<T>(…)**
  - But use the entries where a service must be provided

    TYPE* getDet(IDataProviderSvc* svc,
    const string& location) const
- **Setup options write content**
- **See GaudiExamples/POOLIO/Write**

Match Entities when Reading

- **Using incidents**
  - Whenever a *POOL* file is opened an incident is generated
    - Incident type: "NEW_FILE_RECORD"
      source: FID of new file
  - Algorithm can implement IIncidentHandler interface and handle the incident. Example code snip let:

```
void ReadAlg::handle(const Incident& inc) {
    if ( inc.type() ==  "NEW_FILE_RECORD") {
        string fid = inc.source();  //  The source contains FID
        SmartDataPtr<Counter> cnt(m_recordSvc,fid+"/EvtCount");
        log << "File record: " << n << "=" << cnt->value() << endmsg;
        …
```

Match Entities when Reading

- **Using the persistency triangle**
  - Each dataobject *from file* has a unique address
    This address contains FID
    => string fid = dataobject->registry()->address()->par()[0]

  - This is the fid to match when scanning input files in the file records transient store
  - Helper can be provided if this is main access pattern
- **Example: GaudiExamples/POOLIO/Read**

Possible Extensions

- **Helper to access directory of input file given an object in the event data store**
- **Automatic actions**
  - Combining automatically entities (e.g. adding)
  - Easy said…
  - Generic implementation depends largely on the object model.
  - Cannot be done for “arbitrary” objects
- **…**

Conclusions

- File records can be saved with event data
- Implementation supporting basic features has been done
- Any DataObject can be stored in records store
- More exact use cases, “event model equivalent” do not exist and need to be specified
- Basic functionality is present.
  Extensions can be provided depending on usage