Storing and retrieving Monitoring Results

Helen Hayward

Aims

- Written an AlgTool to aid writing (and retrieving) summary Monitoring data to the ConditionsDB.
- Initially written with SCT in mind.
- ~hhayward/public/11.0.4/AtlasTest/DatabaseTest/IOVDbTestAlg/offline-11-00-41
  - run/UseToolWrite.py
  - src/SCTMonitorConditionsTool.cxx
  - src/SCTConditionsTest.cxx
- Data is written as a COOL folder
  - Warning some technical stuff next!

COOL-Basic Concepts

- COOL implements an **interval of validity(IOV) database**, i.e. objects stored/referenced have an associated start and end time between which they are valid.
- COOL data is stored in **folders**, which are themselves arranged in a hierarchical structure of **foldersets**
  - (in unix terms, folders -> files, and foldersets -> directories),
- Within each folder, several objects of the same type are stored, each with their own IOV(run/event, or as absolute timestamps),
- COOL is optimsed to store and retrieve object(s) associated to a particular time.

COOL structure

- If there are many objects of identical structure (e.g. sct modules):
  - objects in COOL folders can be optionally identified by a **channel number**.
  - Each channel has its own intervals of validity, but all channels can be dealt with together in bulk updates or retrieves.
- COOL implements each folder as a relational database table, with each stored object corresponding to a row in the table. COOL creates columns for the start and end times(and channel number)

COOL payloads

- The remaining **payload** columns are defined by the user. The payload data:
  - can be stored directly in one or more payload columns (**inline** data)
    - columns directly represent the data being stored (e.g. a mixture of float and int columns representing status and parameter information).
  - can be used to **reference** data stored elsewhere.
    - another database table
    - a POOL object reference such as ROOT histograms
  - Can be stored as an **inline BLOB** in the database, i.e. defining the payload to be a large binary or character object (BLOB or **CLOB**) which has internal structure invisible to the COOL database. COOL is then responsible only for storing and retrieving the BLOB/CLOB, and its interpretation is up to the client (e.g. if the CLOB is an XML string which is parsed into a complex C++ structure by the client).

Design of Monitoring Table

- What summary data do we want to write?
  - Per module information
  - Per chip information
  - Per strip information
- We are limited to writing one entry per module.
- Can only write simple objects like integers, floats and strings.
- Solution is to encode an array of chip/defect data as a string (known as CLOB)

Table structure

Channel #	Sample -Size	Barrel_ endcap	Layer_ disk	Phi	Eta	Chip-Data	Defect-List
Int	Int	Int	Int	Int	Int	String	string
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…

•*notation disclaimer: in database language a channel refers to the row of the table …. So in this case a channel = module (confusing I know!)

What do the encoded arrays look like?

- Chip Data, in this example I am assuming we want to record the hit efficiency per chip

```
<Efficiency>
<chip1>1.000000</chip1>
<chip3>11.000000</chip3>
…
</Efficiency>
```

Defect Array

```
<DEFECTLIST>
<DEFECT>
<DefectType>DEAD</DefectType>
<DefectSide>1</DefectSide>
<DefectBeginChannel>100</DefectBeginChannel>
<DefectEndChannel>200</DefectEndChannel>
<hasDefectParameter>False</hasDefectParameter>
<DefectParameter>0.000000</DefectParameter>
</DEFECT>
<DEFECT>
…
</DEFECT>
…
</DEFECTLIST>
```

Using Tool to Write

In initalise:

```
if(StatusCode::SUCCESS != p_toolSvc-
  >retrieveTool("SCTMonitorConditionsTool",m_CondTool))
  {
    m_log<<MSG::ERROR<<"Can not find
  MonitoringConditions Tool"<<endreq;
  }
  m_log<<MSG::DEBUG<<"Found MonitoringConditions
   Tool"<<endreq;

  m_CondTool->newListColl();
```

Using Tool to write (cont).

In execute:

```
//fill the database here
    std::string defectlist;
    std::string EfficiencyTable;

    EfficiencyTable = m_CondTool->addChip(EfficiencyTable,"Efficiency",5,5.0);
    EfficiencyTable = m_CondTool->addChip(EfficiencyTable,"Efficiency",3,11.0);
   ……


    defectlist              = m_CondTool->addDefect(defectlist,1,"DEAD”
                                                    ,100,200,false,0.);
    defectlist              = m_CondTool->addDefect(defectlist,0,"MOSTLYDEAD”
                                                    ,300,400,true,0.95);
   ….

    sc = m_CondTool->createCondObjects_defect(attrListColl,5,1,3,4,25,0,
                                              EfficiencyTable,defectlist);
```

Retrieving information

- What queries would people like?
- Inside COOL itself you cannot search the payload columns (e.g. you cannot ask which channels have a defect)
- So once extracted from COOL, can the AlgTool could answer queries like:
  - For this module return strip range with defect of type "…"
  - For this module and chip number i return value "…"
  - What queries do users want?

Final Database design (Help!)

- Do I design a specific table which all users will use, but not necessarily fill every column in the table?
  - If so I need to know exactly what information needs to be recorded
- Do I focus on having a working example? Which demonstrates easily to users how to add in more columns of the form I already have?
- What functions do people want?