The *New DD*

1. ID Dictionaries
2. Configuration

Vakho Tsulaia (LBNL)

Detector Description Session

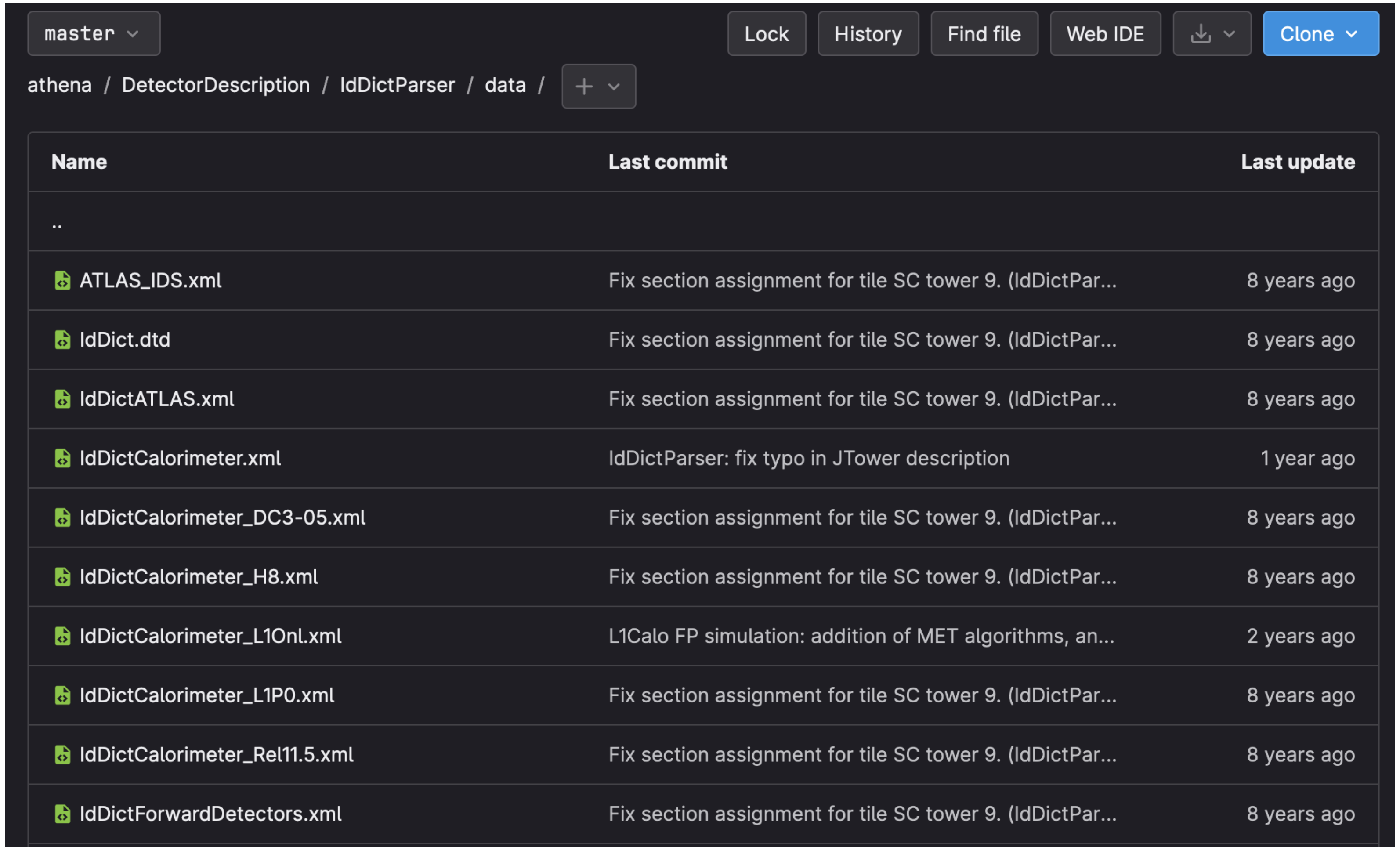
ATLAS S&C Week

June 12, 2023

ID Dictionaries

ID Dictionaries Today (I)

- ID Dictionary files are located in the Athena Git repository
- Location: **`DetectorDescription/IdDictParser/data`**
- 56 dictionary files in total for all ATLAS subsystems
- **Versioning:** if a new dictionary is needed for some subsystem then **a new file** is added to this directory

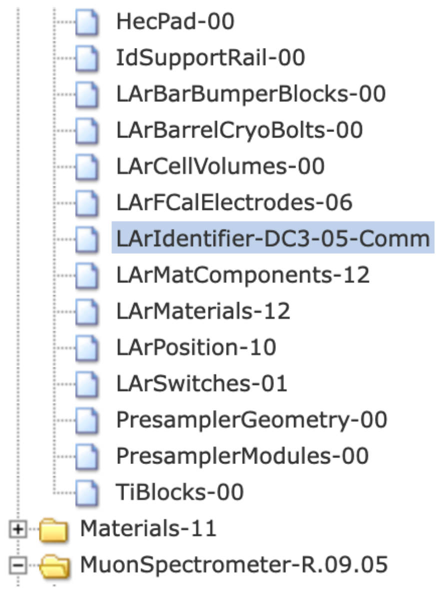


athena / DetectorDescription / IdDictParser / data /

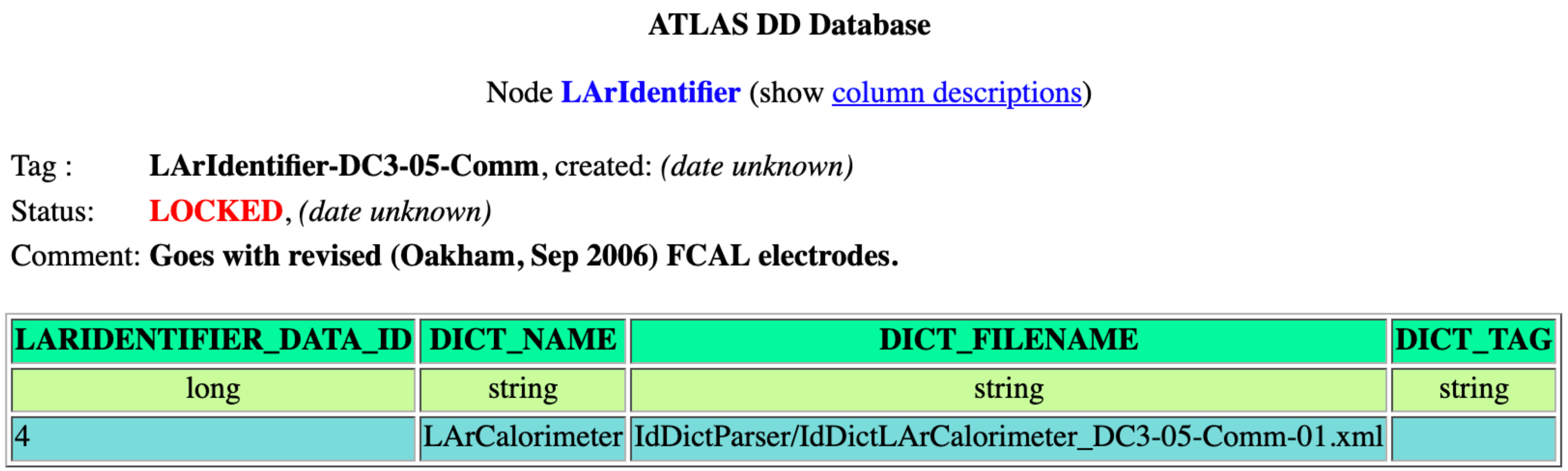
Name	Last commit	Last update
..		
ATLAS_IDS.xml	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago
IdDict.dtd	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago
IdDictATLAS.xml	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago
IdDictCalorimeter.xml	IdDictParser: fix typo in JTower description	1 year ago
IdDictCalorimeter_DC3-05.xml	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago
IdDictCalorimeter_H8.xml	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago
IdDictCalorimeter_L1Onl.xml	L1Calo FP simulation: addition of MET algorithms, an...	2 years ago
IdDictCalorimeter_L1P0.xml	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago
IdDictCalorimeter_Rel11.5.xml	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago
IdDictForwardDetectors.xml	Fix section assignment for tile SC tower 9. (IdDictPar...	8 years ago

ID Dictionaries Today (II)

- ID Dictionary file **names** are **linked with ATLAS tags** in the geometry DB via a bunch of **XXXIdentifier tables** (one table per dictionary)



```
HecPad-00
IdSupportRail-00
LArBarBumperBlocks-00
LArBarrelCryoBolts-00
LArCellVolumes-00
LArFCalElectrodes-06
LArIdentifier-DC3-05-Comm
LArMatComponents-12
LArMaterials-12
LArPosition-10
LArSwitches-01
PresamplerGeometry-00
PresamplerModules-00
TiBlocks-00
Materials-11
MuonSpectrometer-R.09.05
```



ATLAS DD Database

Node **LArIdentifier** (show column descriptions)

Tag : **LArIdentifier-DC3-05-Comm**, created: *(date unknown)*
Status: **LOCKED**, *(date unknown)*
Comment: **Goes with revised (Oakham, Sep 2006) FCAL electrodes.**

LARIDENTIFIER_DATA_ID	DICT_NAME	DICT_FILENAME	DICT_TAG
long	string	string	string
4	LArCalorimeter	IdDictParser/IdDictLArCalorimeter_DC3-05-Comm-01.xml	

ID Dictionaries Today (III)

- In Athena jobs, **IdDictDetDescrCnv::getFileNamesFromTags()** collects the names of all dictionary files for the specific ATLAS version using the **IRDBAccessSvc** interface
- The files are read, and the dictionaries get initialized by the ID dictionary parser

ID Dictionaries in the *New DD* (I)

- **Option #1** (this is how it works now)
- The strategy looks similar to what we have in the old system
- Dictionary file names are specified in the XML “tables”
  - XML “table” is an XML object imported from the Geometry Database using the Oracle-to-XML mechanisms
- The XML “tables” are exported into SQLite tables
  - We call such SQLite tables **Auxiliary Tables**
- **IdDictDetDescrCnv::getFileNamesFromTags()** collects the names of all dictionary files for the specific ATLAS version using the **IRDBAccessSvc** interface
- The files are read, and the dictionaries get initialized by the ID dictionary parser

NB. The green text has been copy-pasted from the previous slide

ID Dictionaries in the *New DD* (II)

- **Option #2**
- Instead of writing **file names** into SQLite tables, **write there the entire XML dictionaries as BLOBs**
- This will require some modifications on the Athena side so that the ID dictionary parser reads dictionaries as strings via the **`IRDBAccessSvc`** interface instead of reading the dictionary files from the disk
- Hopefully, this will not require many complicated changes to the ID dictionary parser code

ID Dictionaries in the *New DD* (III)

- **Option #3**
- Migrate away from XML dictionaries by replacing them with a set of data structures that can be written into SQLite as a one/several relational tables
  - Many years ago, we had an idea/attempt to do such a thing which eventually led to nothing
- This would require major changes in the ID dictionary initialization mechanism
- This would also require rethinking the process of the creation of new dictionaries and fixing the existing ones
  - Over past decades people in ATLAS got used to think about ID dictionaries as XML-s

ID Dictionaries in the *New DD*. Summary

- **Option #1**
  - Write ID Dict file names into SQLite
  - No changes on the Athena side
  - Already works
- **Option #2**
  - Write ID dictionaries as blobs into SQLite
  - Preserve XML structure of the dictionaries
  - Expected to require relatively minor changes on the Athena side
- **Option #3**
  - Replace XML dictionaries with SQLite tables
  - Major change in the ID dictionary initialization procedure

Configuration

Geometry Configuration Today

- The ATLAS **geometry tag** value needs to be set to the **`AtlasVersion`** property of **`GeoModelSvc`**
  - This value can be passed to the job manually (i.e., via python scripts)
  - In the **auto-configuration** mode the job retrieves geometry flag from the **`TagInfo`** in-file metadata using **input file peeking**
  - When running with Job Transforms, the geometry configuration is passed to the job using the **`geometryVersion`** command-line argument
- After obtaining the global geometry version tag, Athena also needs to determine which subsystems are available in the specified geometry version
  - This information is used to avoid creating unnecessary components at configuration
  - The mechanism for detecting available subsystems by geometry version is implemented in **`AutoConfigFlags.py`**
- Everything described on this slide must work after switching to the *New DD* infrastructure

Configuration in the *New DD* (I)

- **Option #1**
- We develop a mechanism for mapping Geometry Tags onto corresponding SQLite database file names
- This requires
  - SQLite databases should be stored in a predefined location on CVMFS
    - For the development and testing purposes the default location can be overridden for a job using an environment variable
  - SQLite database files should be named using ATLAS geometry tags (e.g., **ATLAS-R3S-2021-03-01-00.db**)
- This mechanism should allow the transform to construct a full path to the SQLite file from the geometry version tag
  - Obtained either from the CLI or by peeking into the input file

Configuration in the *New DD* (II)

- **Option #2**
- We write geometry tag value **into SQLite** database
  - The name of the SQLite database file will be passed to Athena/Job Transform
  - The job will peek into SQLite database, read the geometry tag and pass it to GeoModelSvc
- With this strategy we, in principle, can try to get rid of **`geometryVerion`**
  - If desired …
- OTOH, it is not clear how this mechanism can support the retrieval of the geometry tag by peeking into input files (auto-configuration)

Configuration in the *New DD*. Summary

- **Option #1**
  - We provide a mechanism for mapping geometry tags onto SQLite database names
- **Option #2**
  - We write geometry tags into SQLite databases
  - Not clear how we can support geometry version retrieval by input file peeking

- No matter which of the above options we choose, we need to provide a mechanism of peeking into SQLite files at configuration in order to determine the availability of various subsystems in the given SQLite db

Not discussed in this talk …

Constructing valid identifiers inside plugin code

- *What to do if some subsystem needs to construct valid Athena Identifiers in its plugin code?*
- While we could store ID dictionaries in the GeoModelData repository (see slide #7 of this presentation), the code for constructing the Identifiers (ID dictionary parser, ID helpers, etc.) is not available for standalone applications
- Hence, the *New DD* infrastructure does not provide any mechanism for constructing valid identifiers in standalone plugins
- The only way the consistency can be achieved is to do that "by hand", i.e., to build the identifiers inside plugins by some subsystem-specific custom way (e.g., by relying on volume copy numbers or so)