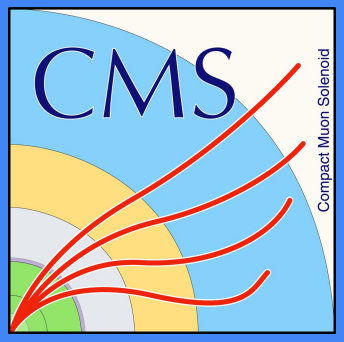
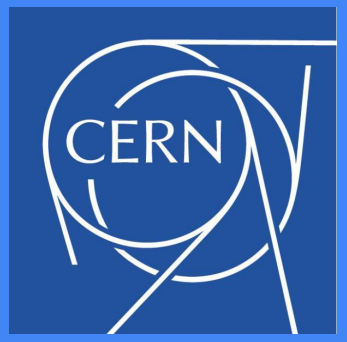




CMS non-event/meta-data

Giacomo Govi, Clemens Lange (CERN)

HSF Data Analysis WG Meeting

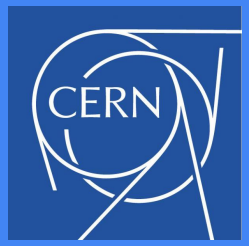
9th May 2019

What is non-event data (in CMS)?

- Everything that is not event data and is needed to produce the analysis result
- Focus on metadata and non-event data used at **analysis level**
- Mind: concrete implementation is analysis-dependent

For most of the non-event data modules exist in CMSSW, but often the data are consumed outside (➜ custom tools)

Fetch-once metadata



3

Book-keeping of files and data sets

- We have a central Data Aggregation System (DAS)
  - Aggregates data from several sources: DBS, PhEDEx, ReqMgr, SiteDB
- Knows details of all centrally and grid-produced samples
  - E.g. production details, location, statistics, …
- Query via web interface or CLI
- Beyond MiniAOD/NanoAOD (e.g. ntuples) no central bookkeeping (modulo hacks)
  - Will change with transition to RUCIO

Sample production cross sections

Several different sources for cross sections:

- Sample-intrinsic information (i.e. generator information)
  - GenXSecAnalyzer (read generator info) or information entered by MC contact
- XSDB (database provided by GEN group)
- Twikis (based on calculation or papers)
- Manual calculation using external tools/calculations

Trigger luminosities

- Provided as a command-line tool (resides on CVMFS, also available for Linux/MacOS standalone)
  - Part of CMS Beam Radiation Instrumentation and Luminosity Worksuite
- Python-based (anaconda) tool
- Only visible (database access) inside CERN (needs tunnel otherwise)

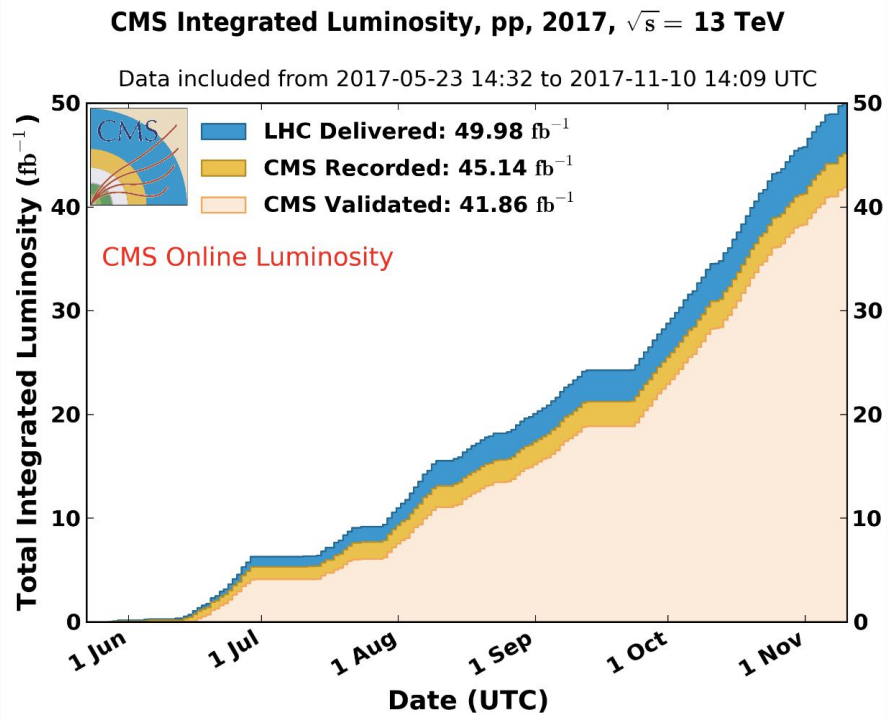
```
brilcalc lumi -b "STABLE BEAMS" --normtag /cvmfs/cms-bril.cern.ch/cms-lumi-pog/Normtags/normtag_PHYSICS.json
-i /afs/cern.ch/cms/CAF/CMSCOMM/COMM_DQM/certification/Collisions17/13TeV/ReReco/Cert_294927-
306462_13TeV_EOY2017ReReco_Collisions17_JSON.txt -u /fb --hltpath "HLT_PFHT1050_v*"
```

```
#Summary:
+------------------+-------+------+-------+------------------+------------------+
| hltpath          | nfill | nrun | ncms  | totdelivered(/fb) | totrecorded(/fb) |
+------------------+-------+------+-------+------------------+------------------+
| HLT_PFHT1050_v10 | 9     | 21   | 5908  | 1.383            | 1.254            |
| HLT_PFHT1050_v11 | 20    | 84   | 23011 | 5.471            | 4.938            |
| HLT_PFHT1050_v12 | 27    | 56   | 23304 | 3.555            | 3.419            |
| HLT_PFHT1050_v13 | 18    | 47   | 19043 | 2.959            | 2.862            |
| HLT_PFHT1050_v14 | 67    | 144  | 90816 | 21.632           | 20.596           |
| HLT_PFHT1050_v15 | 6     | 27   | 14589 | 2.954            | 2.802            |
| HLT_PFHT1050_v16 | 2     | 13   | 4469  | 0.901            | 0.862            |
| HLT_PFHT1050_v7  | 2     | 3    | 1775  | 0.247            | 0.238            |
| HLT_PFHT1050_v8  | 11    | 43   | 14172 | 2.794            | 2.685            |
| HLT_PFHT1050_v9  | 13    | 36   | 8349  | 2.007            | 1.870            |
+------------------+-------+------+-------+------------------+------------------+
```

Data quality

Data quality provided for different analysis purposes in JSON format (cf. ATLAS GoodRunsLists in XML format):

- DCS JSON
- Muon JSON
- Golden JSON
  (used by most analyses)

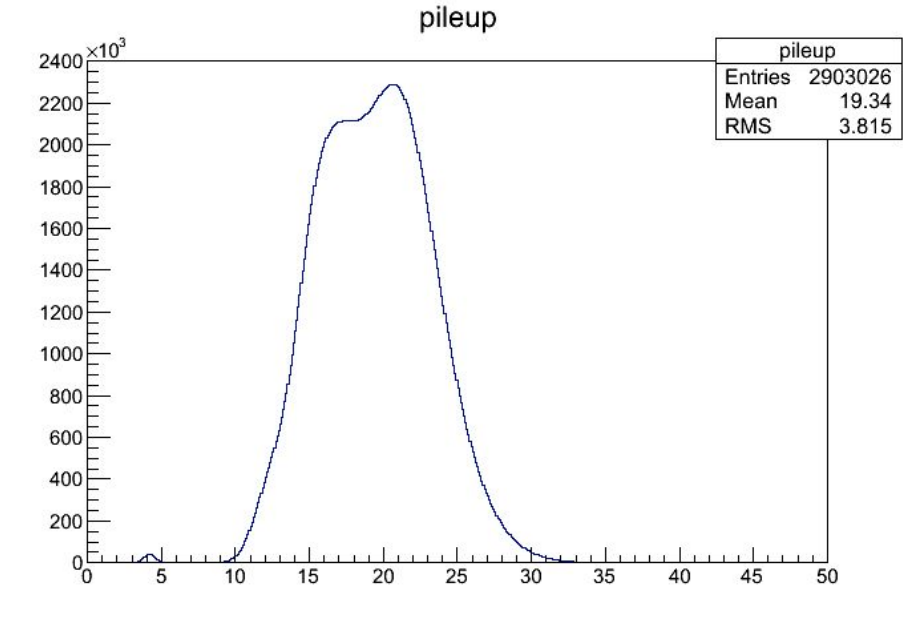


Filter on good runs and lumi-sections

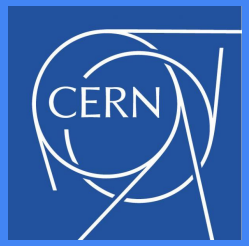
- Mostly used in last CMSSW-based analysis step or custom analysis code
- Files reside on AFS

Pileup reweighting

- For each MC campaign, have central simulation pileup (PU) profile
- Extract data PU profile using BRIL Worksuite for choice of triggers
- Reweight on event/sample basis (event weights)
  - Again, possible in CMSSW, and analysts developed custom tools for use outside



CMS Conditions Database



9

Conditions

- Scope
  - Non-event information concerning the detector subsystems
  - Varying with time
  - Required in the primary processing workflows: HLT, Express, Prompt, Reco
- Data categories
  - Status and Configuration for Detectors, Trigger menu
  - Detector Calibrations
  - Run Information, Beam and Luminosity information
  - Parameters for physics algorithms
- Technologies
  - Oracle for master DB storage
  - Frontier for read-only access in standard processing

Generally only accessible via CMSSW (see later)

Condition Data Model

- Payload
  - Values exclusively consumed within the CMSSW Framework
  - Uniquely described by a C++ class (typically header+param array)
  - Serialized into binary array (boost technology), stored as a BLOB
- Interval of validity (IOV)
  - Time information for the lookup by event
  - Run number, Lumisection and Timestamp granularity
  - Defined only by ‘since time’ (open IOV)
- Tag
  - Label identifying/categorizing a specific set of IOVs
  - Defined for a specific Payload type
- Global Tag
  - A consistent Set of Tags targeted to a specific workflow/scenario
  - Mind: no semantic versioning but custom naming

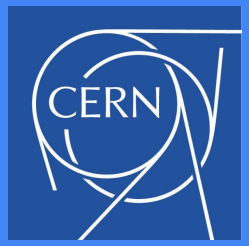
Updating/Customizing conditions

- Data taking use case
  - Tags involved in GTs for standard workflows can be updated with payloads/iovs for new run ongoing
  - No need to create new GT
- Reprocessing
  - New Payload/IOV sets created while re-processing conditions will go in new Tags
  - New tags are submitted to GT queues for the standard workflows/scenarios
  - New GTs are created out of snapshots from the queues, after validation
- Customisation
  - One or more Tags in a standard GT can be ‘overwritten’ with Tags stored in local files, owned by the user.

Condition database: analysis use-cases

- Workarounds with the present infrastructure
  - Local processing, fully stand-alone
    - Require the export into files of the concerned GT
    - Currently only supported in RDBMS technology (sqlite)
    - Accounts for a total file size up to 2-3 Gbytes
  - Payload exports into text files
    - Individual payloads can be selected and fetched from the DB
    - A command line tool allow to de-serialize and dump into XML format
- Developments to be planned
  - Need to define the requirements with the customers
  - Going towards fully readable exports from Condition DB
    - Data model to be adapted to real use cases
    - Metadata (IOV, Tag, GT) handled with files and folder structures
    - Payloads serialized in JSON

Update-often metadata



14

Use of in-file metadata

- CMSSW data format contains provenance information
  - E.g. production details, cross sections
- Usually very little use of in-file metadata beyond that
  - Again, mostly only accessible from within CMSSW

If this was more flexible, could store lots of useful (provenance/workflow) information e.g. also in ntuples

Physics-object corrections

- Corrections applied to jets, leptons, tracks etc. to account for differences in simulation and data
- Provided/derived by Physics Object Groups (POGs)

- So-called POG samples are produced before general production/reconstructions campaigns for validation and to have corrections available asap
- Nevertheless, corrections mostly only available after these campaigns (i.e. not part of global tag for reconstruction)
- Analysts need to apply latest corrections on-the-fly
  - Note: included in NanoAOD processing
  - This similarly applies to efficiency calculations

Technologies serving the non-event/meta-data

Missing standardised way of providing corrections:

- Databases ➜ lumi, conditions
- TXT files ➜ jet energy corrections
- ROOT files with histograms ➜ muon/electron efficiencies and scale factors
- JSON files ➜ good runs/lumi sections
- CSV files ➜ b-tagging

In most cases code snippets or libraries exist for use in analysis

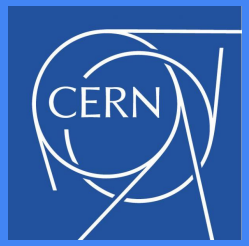
- Validation of actual implementation is left to the analyst

Versioning metadata

- Metadata in conditions database versioned via tags and IOVs
- Physics-object metadata is mostly versioned via filename
  - Unfortunate side-effect: bloats analysis code
  - No common approach followed

This similarly applied for metadata for different data-taking periods

Wishlist



19

Common metadata interface?

Providing a common interface for non-event/meta-data would greatly simplify analysis work

- Currently, each group is re-inventing the wheel
  - Need to learn/use different tool for each object
- CLIs could be streamlined
  - Standalone binaries for different platforms (golang?)
  - Common authentication system
  - JSON response (or similar)
- Provide web interfaces/API
  - Again, each group seems to develop something custom
- Write as much metadata as possible into ROOT files
  - Provide easy way of doing that (maybe exists already?)
- Clear candidate for linked data ➜ GraphDB?
  - Could aggregate all information in one single place