CMS conditions database web application service

Katarzyna Maria Dziedziniewicz[1], **Domenico Giordano**[1,2], **Vincenzo Innocente**[1], **Anne-Catherine Le Bihan**[3], **Antonio Pierro**[2], **Zhen XIE**[4]

[1] CERN
[2] INFN-Bari - Bari University, Via Orabona 4, Bari 70126, Italy
[3] Institut Pluridisciplinaire Hubert Curien. Strasbourg
[4] Princeton University

E-mail: Katarzyna.Maria.Dziedziniewicz@cern.ch, domenico.giordano@cern.ch, vincenzo.innocente@cern.ch, alebihan@mail.cern.ch, antonio.pierro@cern.chi, Zhen.Xie@cern.ch

Abstract. The web application service is part of the condition database system of the CMS experiment.

The application server is built upon condition python API in the CMS offline software framework and serves applications and users not involved in the event-processing.

The main client of the application server is the condition database web GUI, which currently exposes three main services: the Tag Browser, the Global Tag, the Historic Chart condition date in terms of their version (TAG) and the interval of validity (IOV).

The global tag component is used by physicists to inspect the organization of the tags in a given data taking or data production, while production managers use the web service to produce such tag hierarchy. The History chart plotting service creates dynamic summary and distribution charts of the payload data in the database. Fast graphical overview of different information greatly contributes in monitoring and validating the calibration data stored in the condition database.

1. Introduction

In order to reconstruct the physical events of CMS experiment[1], the physicists use quantities not changing from event to event and for that reason they are called *Not-Event Data*. The *Not-Event Data*, also called *Condition Data*, produced by CMS detector, are stored in the CMS condition database and can be roughly divided in three groups:

- *Configuration data*: the data needed to bring the detector in running mode;
- *Detector state data*: the data from any detector subsystem describing its state;
- *Calibration data*: the data describing the calibration and alignment of the single pieces of different subdetector.

Calibration, Configuration and Detector state data, coming from the sub-detectors' computers, from network devices and from different sources, are packed as C++ objects and moved to the Online condition database, hence using ORACLE streaming to the Offline condition database called *ORCOFF* (Offline Reconstruction Condition DB Offiline subset), which is the main *condition database* for the CMS T0 (tier 0)[2].

Each conditions data, which is stored in the offline database as POOL-ORA[1] objects, is also called *payload object*. Each payload object is indexed by its interval of validity (IOV), while the payload objects themselves do not contain any time validity related information. A IOV sequence can be indexed with another metadata information, know as TAG, which is a human-readable label describing the version of condition data. The IOV index and the TAGs can be deleted and recreated since it is independent of the data object it points to.

When processing large quantity of calibration/alignment data, the production manager needs to organize collections of IOV TAGs coming from different data sources, i.e. databases. In order to help the user to manage these collections of IOV and TAGs the concept of *Global TAGs* was introduced. It defines the scope of a collection of basic IOV TAGs. An ensemble of *Global TAGs* can be organized in a hierarchy. Typically *Global TAGs* and the *TAG hierarchy* are used by the production manager to organize collections of basic IOV TAGs collected from different sources.

In this scenario, the *CMS condition database web application server* is used for back-end purpose to serve application and users for accessing, monitoring and handling these objects.

The main client of this application server is the *condition database web GUI* that exposes three main services: *IOV Tag browsing*, *Global tag management* and *Historical data quality monitoring*.

The organization of this paper is the following: In section 2 the architecture of this web application service is described , in the following three sections, the *IOV Tag Browsing*, *Global Tag Management* and the *History Data Quality Monitoring* are described; finally last section sums up the conclusions.

2. Architecture of CMS conditions database web application service

The architecture of *CMS conditions database web application service* can be designed in two distinct components: *the CMS Web Frontend* and the *Web CondDb Backend* which is made of the *CMS Web Interface* and the *CMS Application Server* (fig. 1).

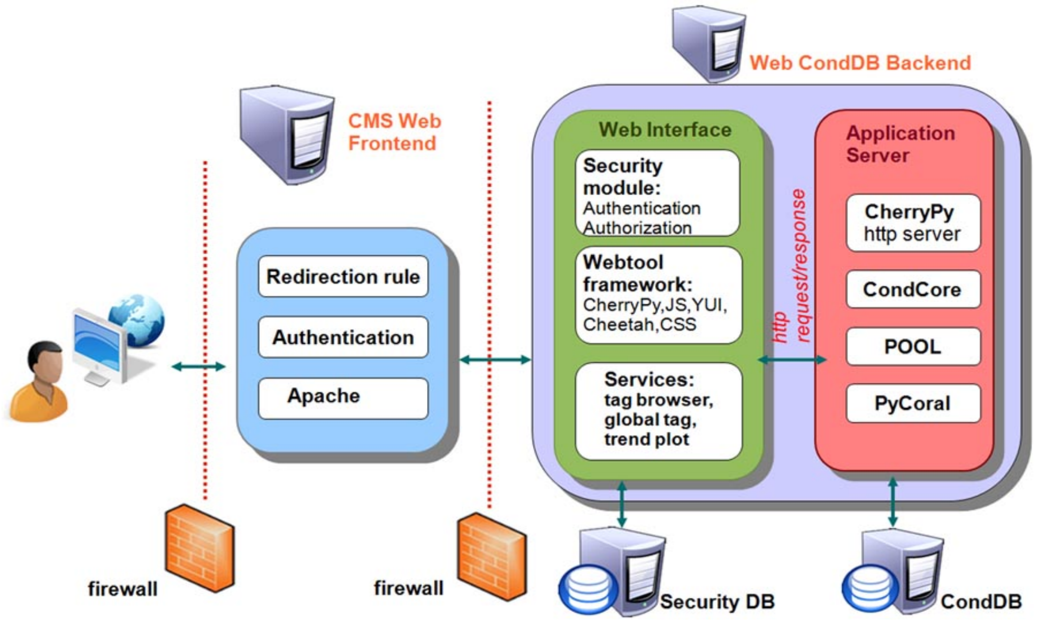


Figure 1. Architecture of CMS conditions database web application service.

[1] POOL provides object relational mapping between the C++ clients and various database technologies.

2.1. CMS Web Frontend

The *front end servers* are accessible to the outside world, while the *Web ConDB Backend* machines are firewalled off from remote access. The CMS Web Frontend provides a consistent interface to logging, by showing different pages according to the role the user has inside the CMS collaboration, thanks to *redirection-rule* attribute.

This component is shared by all CMS web applications. For further details and information about the CMS Web Fronted refer to *CMS Offline Web Tools*[3].

2.2. Web CondDB Backend

The *Web CondDB Backend* consists of two layers: the *web service* and the *application server*. The *web service* forwards the user HTTP requests to the *application server*.

All the *web services* rely heavily on backend databases Oracle (Security database and Conditions databases) and layers CMSSW (CondCore), POOL, PyCoral[2] and CherryPy[3]. Database access is made using POOL and PyCoral that generate SQL queries from CMSSW C++ objects[4].

The *HTTP request* goes through *Application server* that converts the HTTP request in a script run by CMSSW framework and sends that to the oracle server. The results go back to the *Web Service* in the reverse direction, into an HTTP formatted stream. Finally the *Web Service* manages the *HTTP response* displaying the data to the end-users in required format (lists, tables or chart reports).

The *application server* and the *web service* may be deployed on different hosts. They are deployed with distinct *RPM packages*[4].

2.3. CMS Web Interface

The design of the *CMS Web Interface* is based on a the central class which is called "*Controller*" from which these three *web service* derive from.

The webtools framework relies on a very simple plug-in mechanism, that allows a separate development of the components, without any central component list.

When the *application server* is run, the framework will look up in each entry of python path for directories called respectively "Modules", "Controllers" or "Applications". If any is found it will attempt to (python) import every file in those directory.

The *CMS Web Interface* is completely implemented in python. The reason of web development using python was driven by strong support for integration with other languages and tools coming with extensive standard libraries. Moreover Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, high level dynamic data types, and dynamic typing[5]. Python had a great set of tools to accommodate our tasks. Among a large number of possibilities we choose the following components:

- *CherryPy*, an object-oriented HTTP framework in Python, with flexible configuration and extension.
- *Cheetah template framework*[6], written in python for presentation layer of web pages.

[2] The PyCoral package provides a python module named coral, exposing the user-level interfaces of CORAL in a native python style. Its implementation is based on the C++ CORAL libraries.
[3] CherryPy: a pythonic, object-oriented HTTP framework.
[4] An RPM package is a single file containing all of the components necessary to install one feature, such as a software application, a group of related utilities, a driver, or a set or fonts or artwork files.
[5] A dynamic type system is a mechanism by which a compiler generates code to keep track of the sort of data (incidentally still called "type") used by the program.
[6] Cheetah is an open source template engine and code generation tool

- *Python decorator*[7] *for Authentication and Authorisation.* This python-decorator can be used to create a web site that behaves differently according to whether a user is authorized or not, and according to his/her role.The task that can be performed are:
  - Authentication; it means a user is known to the Security database (e.g. have a hypernews account, or are listed in SiteDB for some other reason)
  - Authorisation; it means that a given user is known to the database and has some role associated to a group or site (for example a Production Operator in the PRODREQUEST region, or the Data Manager).
- *Boost-Python*, a C++ library which enables seamless interoperability between C++ and the Python programming language.
- *PyOFC2*(Python Open Flash Chart 2), a python library to generate *JSON*[8] data that can be consumed by Adobe Flash Player[9] Charts.

The **front-end** of this python-based framework is a GUI that implements technologies, like :

- *Cascading Style Sheets* (CSS)[10] technologies makes web page flexible, user friendly and most important easy to develop and change the design of the web page. so that a user feels comfortable with a standard style for all web pages.
- *Cache technologies* for instant querying to Data Base if up-to-date data are not needed.
- *Yahoo Interface Library* YUI; a set of utilities and controls, written in JavaScript, for building richly interactive web applications using techniques such as DOM scripting, DHTML; where possible these are reused to provide identical functionality across different components, so that a user learns a single interface to all web tools.
- *jQuery* a fast and concise JavaScript Library that simplifies event handling, animating, and Ajaj interactions for rapid web development. One example among many Javascript Library is (*Javascript validation*) used to assist the users to fill the forms out correctly and simplify the process of filling out.

2.4. Application Server

The *Application Server* is a linux daemon that delivers conditions database query results from *CondCore software* (selectively packaged CMSSW) on request.

The **back-end** of this python-based framework contains a *python plug-in* binding a template class that wraps the *condition objects* and declare a set of methods to be used to inspect that objects. This approach has several advantages, such as being independent from schema changes, providing a common syntax and behaviour to inspect *payload class* using generic code and being transparent to the DB back-end.

In the following paragraphs, we will describe the web services that can exploit, independently, the application server to inspect the Condition Objects through HTTP requests.

3. IOV TAG Browsing Service

The *IOV TAG Browsing Service* allows to discover the version (TAG) and the interval of validity (IOV) of each of set of conditions.

[7] A Python decorator is a specific change to the Python syntax that allows us to more conveniently alter functions and methods (and possibly classes in a future version).

[8] JSON (JavaScript Object Notation) is a lightweight data-interchange format.

[9] Adobe Flash Player is the standard for delivering high-impact, rich Web content. Designs, animation, and application user interfaces are deployed immediately across all browsers and platforms, attracting and engaging users with a rich Web experience.

[10] Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colours, spacing) to Web documents.

A basic IOV tag is produced by calibration or alignment process which uniquely identifies an IOV sequence. It is the lowest level entry point to retrieve time-dependent data from the database. It is stored together with the data which can be used directly to retrieve calibration and alignment data. The IOV tag concept is explained here. In the tag DB, a collection of basic IOV tags is stored in the tag inventory. Each entry in the basic tag inventory contains the following summary information: basic tag name, source database(in the format of connection string), EventSetup record name, payload object name, record run time label name, timetype of the record. This service allows a user to view the tags and Interval of Validity (IOV) information concerning the conditions data; it is like a web interface to CMSSW command line tools; effectively a web interface to CMSSW command line tools. The *IOV Tag Browsing Service* consists of 4 parts:

- *The menu header*; this menu is common to all web service in *CMS Condition Database* and allows to navigate with the same style between all web services. The tabs for the condition database service (CondDB) are blue, while the other CMS services are greyed out (Figure2).

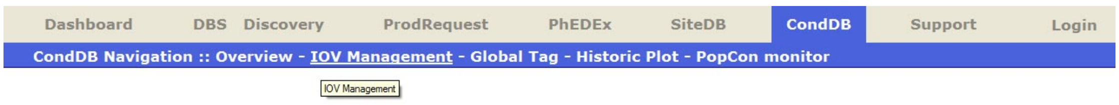
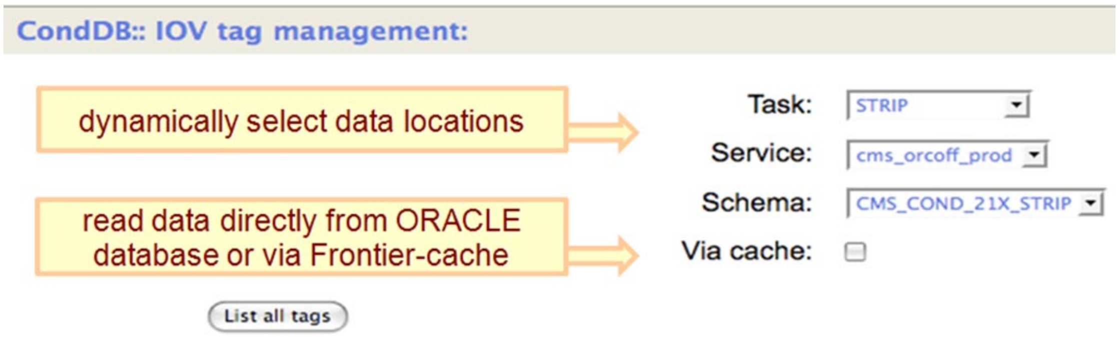

Dashboard
DBS Discovery
ProdRequest
PhEDEx
SiteDB
CondDB
Support
Login
CondDB Navigation :: Overview - IOV Management - Global Tag - Historic Plot - PopCon monitor
IOV Management

Figure 2. Horizontal tab menu used to navigate between different web service

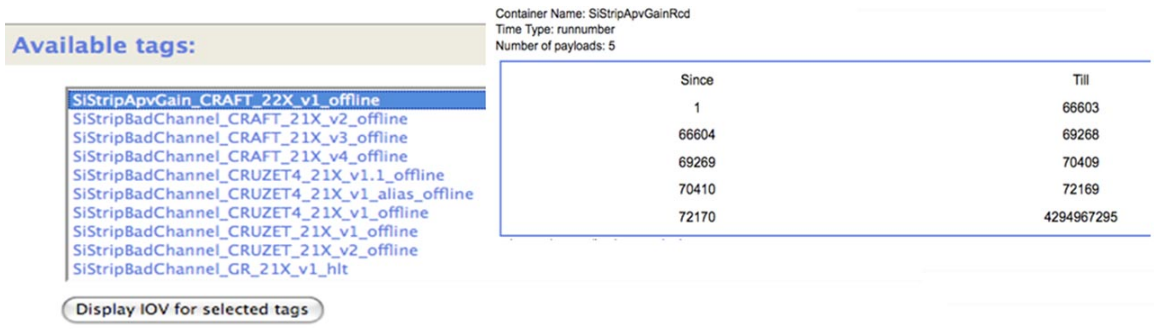
- *CondDB::IOV tag management* section; it is a menu-driven approach which guides users through their selection of tag: e.g. Task (Detector), Service (service name used to connect to the remote database), Schema (account's database). To the user is given the possibility to choose to read data directly from ORACLE database or via Frontier-cache[5] that provides high performance data access, since given object may be used by thousands of users and by decree, data in condition IOV cannot change. So caching such information provides significant performance gains (Figure 3).


CondDB:: IOV tag management:
dynamically select data locations
read data directly from ORACLE database or via Frontier-cache
List all tags

Task: STRIP
Service: cms_orcoff_prod
Schema: CMS_COND_21X_STRIP
Via cache: ☐

Figure 3. IOV/TAG Browsing Service section

- *Available tags* section; it allows a user to browse condition data in terms of IOV and the version (TAG). In details, clicking on a TAG will allow user to view IOV in terms of two integer values: since and till (Figure 4).
- *Export selected data with optional parameters* section; it allows users to export the selected data from production databases to private SQLite file filtering the data, choosing the *TAG* name and the IOV in the form of two integers (the first since and the last till). (Figure 5).



Available tags:

SiStripApvGain_CRAFT_22X_v1_offline
SiStripBadChannel_CRAFT_21X_v2_offline
SiStripBadChannel_CRAFT_21X_v3_offline
SiStripBadChannel_CRAFT_21X_v4_offline
SiStripBadChannel_CRUZET4_21X_v1.1_offline
SiStripBadChannel_CRUZET4_21X_v1_alias_offline
SiStripBadChannel_CRUZET4_21X_v1_offline
SiStripBadChannel_CRUZET_21X_v1_offline
SiStripBadChannel_CRUZET_21X_v2_offline
SiStripBadChannel_GR_21X_v1_hlt

Display IOV for selected tags

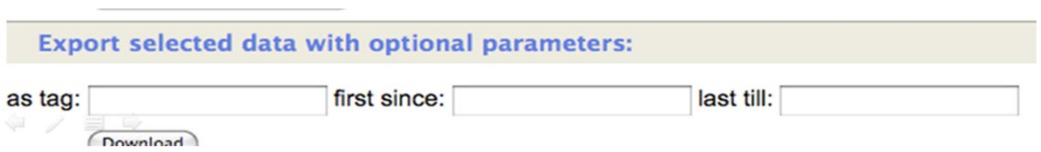
Container Name: SiStripApvGainRcd
Time Type: runnumber
Number of payloads: 5

Since	Till
1	66603
66604	69268
69269	70409
70410	72169
72170	4294967295

Figure 4. IOV Tag Browsing Service. For each TAG the user can display IOV in terms of two integer values: since and till

4. Global Tag Management Service

The global tag component is used by *physicists* to inspect the organization of the tags in a given data taking or data production while *production managers* use the web service to produce such tag hierarchy.



Export selected data with optional parameters:

as tag: first since: last till:

Download

Figure 5. This section of *IOVTag Browsing Service* allows users to export selected data with optional parameters. The file can be exported by clicking on download button below.

The GUI6 allows to select database account and for each database account it is possible to browse three section.

- *Tree list*: It shows the list of tag trees for the selected account.
- *Tag inventory*: it shows IOV tags for a selected account in a *Select multiple form field* with an internal scroll bar.
- *Tag tree*: it can be used in two ways: to select a tag tree and browse it and for the *production managers* to edit the tag tree and its node. In details, the actions that *production managers* can perform are: add child node, edit node label, delete node, delete node with children, view TAG info and view IOV info.

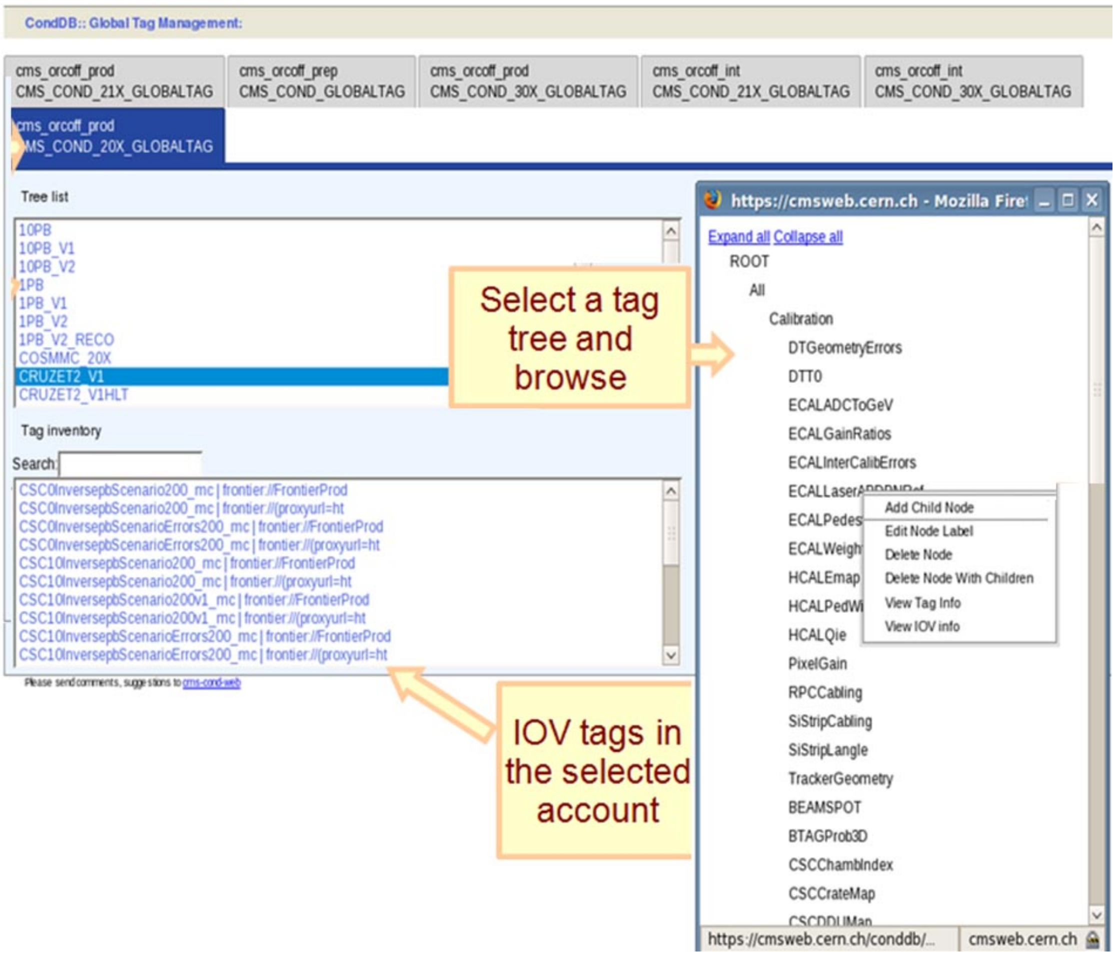
5. History Data Quality Monitoring service

History DQM is designed to follow the time evolution of relevant quantities monitored by the DQM tasks. The service creates dynamic summaries and distribution charts of these quantities stored as condition objects. This tool provides the monitoring monitoring of the detector performance with reference to time and gives prompt feedback about the stability of data taking.

5.1. Architecture and design of the history DQM tool

The main steps of history DQM are sketched on Figure 7:

- extraction of summary information from DQM and population of the condition database;
- extraction of the summary information from the condition database;
- creation and visualisation of the trend charts by means of the application service described above.


CondDB:: Global Tag Management:
cms_orcoff_prod CMS_COND_21X_GLOBALTAG
cms_orcoff_prep CMS_COND_GLOBALTAG
cms_orcoff_prod CMS_COND_30X_GLOBALTAG
cms_orcoff_int CMS_COND_21X_GLOBALTAG
cms_orcoff_int CMS_COND_30X_GLOBALTAG
cms_orcoff_prod CMS_COND_20X_GLOBALTAG
Tree list
Tag inventory
Search:
Select a tag tree and browse
IOV tags in the selected account
https://cmsweb.cern.ch - Mozilla Firefox
Expand all Collapse all
Add Child Node
Edit Node Label
Delete Node
Delete Node With Children
View Tag Info
View IOV info
https://cmsweb.cern.ch/conddb/...
cmsweb.cern.ch

10PB
10PB_V1
10PB_V2
1PB
1PB_V1
1PB_V2
1PB_V2_RECO
COSMMC_20X
CRUZET2_V1
CRUZET2_V1HLT

CSC0InversepbScenario200_mc | frontier://FrontierProd
CSC0InversepbScenario200_mc | frontier://(proxyurl=ht
CSC0InversepbScenarioErrors200_mc | frontier://FrontierProd
CSC0InversepbScenarioErrors200_mc | frontier://(proxyurl=ht
CSC10InversepbScenario200_mc | frontier://FrontierProd
CSC10InversepbScenario200_mc | frontier://(proxyurl=ht
CSC10InversepbScenario200v1_mc | frontier://FrontierProd
CSC10InversepbScenario200v1_mc | frontier://(proxyurl=ht
CSC10InversepbScenarioErrors200_mc | frontier://FrontierProd
CSC10InversepbScenarioErrors200_mc | frontier://(proxyurl=ht

Please send comments, suggestions to cms-cond-web

ROOT
All
Calibration
DTGeometryErrors
DTT0
ECALADCToGeV
ECALGainRatios
ECALInterCalibErrors
ECALLaserAPDPNRef
ECALPedes
ECALWeigh
HCALEmap
HCALPedW
HCALQie
PixelGain
RPCCabling
SiStripCabling
SiStripLangle
TrackerGeometry
BEAMSPOT
BTAGProb3D
CSCChambIndex
CSCCrateMap
CSCDDUMap

Figure 6. GUI of Global Tag Management Service

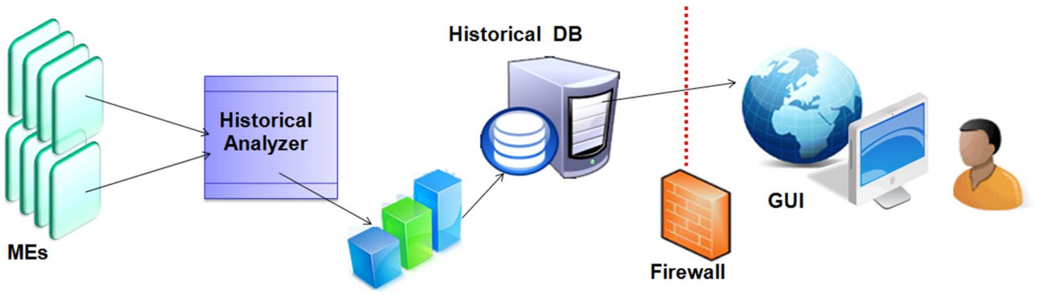
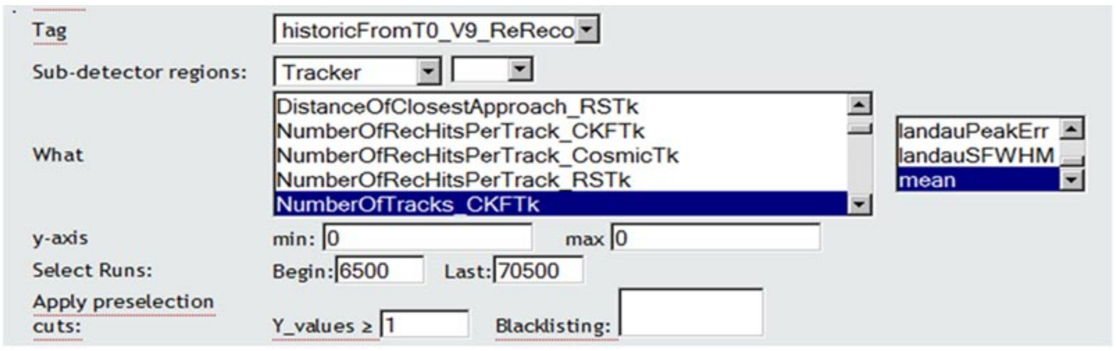


Figure 7. Architecture and design of History DQM

5.2. The web form of Historical Data Quality Monioring

The web form of Historical Data Quality Monitoring provides access to the summary quantities extracted from the Condition DB.

The *dynamic web form* integrated in the *web interface* (fig. 8), guides and aides the user to fill the fields required to get trend charts, checking the consistency of given arguments (e.g. it warns the user if some information is missing or incorrect), and suggesting the quantities available for inspection. For example, it prompts the corresponding mathematical quantities



Tag	historicFromT0_V9_ReReco	
Sub-detector regions:	Tracker	
What	DistanceOfClosestApproach_RSTk NumberOfRecHitsPerTrack_CKFTk NumberOfRecHitsPerTrack_CosmicTk NumberOfRecHitsPerTrack_RSTk NumberOfTracks_CKFTk	landauPeakErr landauSFWHM mean
y-axis	min: 0	max 0
Select Runs:	Begin: 6500	Last: 70500
Apply preselection cuts:	Y_values ≥ 1	Blacklisting:

Figure 8. The Web form of Historical Data Quality Monitoring

(mean, RMS, Landau, Gauss, etc.) for each physical quantities (number of tracks, number of hits, etc.).

For a given set of summary data, through the web form (fig. 8), the user can query the following information:

(i) the full collection of timestamps;

(ii) the quantities which have been stored in the database;

(iii) the corresponding sub-detector regions for which the stored information are available. Dynamic drop down list helps the user to navigate through all the Subdetector regions into a dynamic menu tree.

As points are concerned (ii) and (iii), each sub detector could have different quantities. These quantities are retrieved by means of python plugins which provide a flexible way to inspect the data stored in database by calling methods that access interfaces supported by CMSSW[6].

Besides the basic retrieval of the summary informations from the database, are the possibility to apply preselection cuts on the stored quantities and to remove undesired entries by mean of black list field.

5.3. Trend plots

For the graphics report, a web-based plug-in has been chosen, since it is more efficient and less time consuming than a server-side application like ROOT-CERN[11].

The ROOT CERN application works well locally, but it is not so good for remote operations like a web server.

As the web-based plug-in chart is concerned, we chose *PyOFC2* (Python Open Flash Chart 2) for three reasons:

- it is entirely based on Python, like our web service;
- it can read *JSON* data source, generally smaller than a XML document, and works faster with it;
- as the user moves the mouse over the chart, it displays the values for each line series along vertical tracking line, allowing the users to quickly identify the exact value (run and time).

As an example of chart, the figure 9 shows the average number of tracks in the CMS Tracker as a function of time created during the Cosmic Data taking of CMS in Autumn of 2008.

[11] ROOT is an object-oriented program and library developed by CERN and designed for particle physics data analysis.

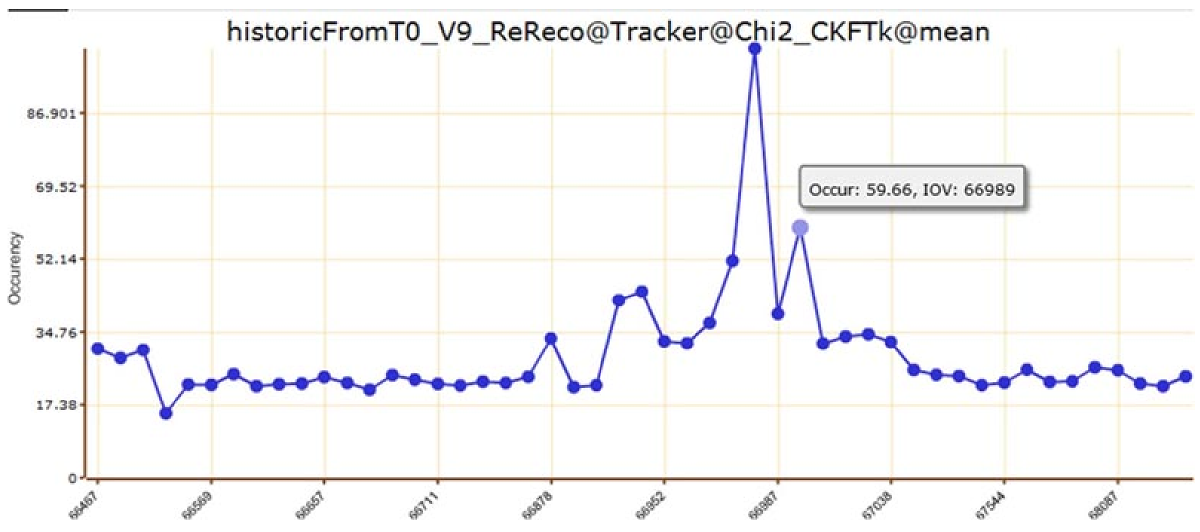


Figure 9. Mean number of reconstructed tracks per event as a function of *run-numbers* (progressive number given to a set of contiguous Physics events, caraterised by the same conditions for operation and detector status). This chart is an example of trend plot extracted from the Cosmic Data taking of CMS in Autumn of 2008. With the help of the mouse, users can interact directly with the chart. Users can point the cursor to the part of chart and see the information about the run.

6. Conclusion

We discussed the *Conditions database web application service*, for the CMS experiment, using emerging techniques commonly called *web 2.0*.

The use of standard software, such as *Cheetah template* and various python modules, makes the system reliable, highly configurable and easily maintainable.

The History-DQM web service has recently been adopted by the CMS DQM System to provide history for both online and offline monitoring during the last Cosmic Data taking of CMS in Autumn of 2008.

In the future these web services will become more sophisticated, incorporating role based views of the production system and more complex queries and plots, supporting the needs of the whole CMS DQM community. The tool must quickly provide access to hundreds of thousands of histograms, and be tuned to each user's needs. The average GUI HTTP response time is around 10 seconds without cache and usually less 1 second with cache, a performance considered acceptable to the practical needs of the experiment.

The development cycle of these web services pointed out the benefits coming from the use of both python and template engine. However since a fast turnaround with user requirements is necessary, especially for a huge collaboration like CMS, their interfaces could have some modification.

References

[1] The CMS Collaboration. CMS Physics TDR, Volume I: Detector Performance and Software. Technical Report CERN-LHCC-2006-001; CMSTDR-008-1, CERN, Geneva, 2006.
[2] First experience in operating the population of the condition database for the CMS experiment - CHEP09.
[3] CMS Offline Web Tools - CHEP 07 J. Physics: Conference Series 119 (2008) 082007.
[4] CMS Offline Conditions Framework and Services - CHEP 09.
[5] CMS Conditions Data access using FronNTier. J. Phys.: Conference Series 119 (2008) 072007.
[6] CMS Computing TDR, CERN-LHCC-2005-023, ¡http://cdsweb.cern.ch/record/838359¿ 20 June 2005.

[7] Cond DB web application service. `https://twiki.cern.ch/twiki/bin/view/CMS/CondDBWebDeployment`.