The Data Quality Monitoring Software for the CMS experiment at the LHC

M Rovere\(^1\) on behalf of the CMS collaboration

\(^1\)CERN, Geneva, Switzerland
E-mail: marco.rovere@cern.ch

Abstract.

The Data Quality Monitoring (DQM) Software is a central tool in the CMS experiment. Its flexibility allows for integration in several key environments: Online, for real-time detector monitoring; Offline, for the final, fine-grained data analysis and certification; Release-Validation, to constantly validate the functionality and the performance of the reconstruction software; in Monte Carlo productions. Since the end of data taking at a center of mass energy of 8 TeV, the environment in which the DQM lives has undergone fundamental changes. In turn, the DQM system has made significant upgrades in many areas to respond to not only the changes in infrastructure, but also the growing specialized needs of the collaboration with an emphasis on more sophisticated methods for evaluating data quality, as well as advancing the DQM system to provide quality assessments of various Monte Carlo simulations versus data distributions, monitoring changes in physical effects due to modifications of algorithms or framework, and enabling regression modeling for long-term effects for the CMS detector. The central tool to deliver Data Quality information is an interactive web site for browsing data quality histograms (DQMGUI). In this contribution the usage of the DQM Software in the different environments and its integration in the CMS Reconstruction Software Framework (CMSSW) and in all production workflows are presented, with emphasis on recent developments and improvement in advance of the LHC restart at 13 TeV. The main technical challenges and the adopted solutions to them will be also discussed with emphasis on functionality and long-term robustness.

1. Introduction

The Compact Muon Solenoid (CMS)[1] is a multi-purpose detector at the Large Hadron Collider[2] at CERN. Data Quality Monitoring (DQM) is a critical component for the detector and operation efficiency and for a reliable certification of the recorded data. In the following sections the implementation of the DQM Framework during RunI will be described, with particular emphasis to its connection to other components of the CMS experiment. Also, the improvements to the DQM framework in the face of the fundamental changes that happened during the Long Shutdown I in the surrounding components (Online/DAQ and Offline/CMSSW) will be thoroughly analyzed, together with their implications in the partial redesign of the framework.

2. The DQM Framework

The CMS collaboration has adopted a single end-to-end DQM framework to cover all possible use cases[3], as illustrated in Figure 1. The DQM framework proved to be extremely flexible during RunI and has been used in almost all areas of the CMS experiment. In online, it performed a
Live monitoring of the status of the various detectors during data taking and to give the online crew the possibility to spot problems as soon as they appeared. In offline, during the prompt reconstruction performed at Tier-0, it provided the final quality flags used to perform the data certification. During the release validation, which is a central workflow that is regularly run on a bi-weekly basis, it assessed the goodness of the new reconstruction algorithms and of high-level physics objects definitions that are regularly integrated in the CMS software release framework (CMSSW). In the so called prompt calibration loop[4], it was used to compute and validate the most up-to-date set of conditions and calibrations that will be subsequently used during the prompt reconstruction. Finally, during the early stages of Monte Carlo production, it monitored the quality of the samples that will then be used for publications.

![Diagram](image)

**Figure 1.** Usage of the DQM framework in the CMS experiment.

The main components of the DQM framework are illustrated in Figure 2. The key elements are:

- the *DQMStore*, which is a unique block of shared memory, whose validity spans the entire lifetime of the process, that internally holds all the DQM information
- the *MonitorElement* (ME), which is the building block of all the data quality monitoring information in CMS. It is a wrapper around ROOT[5] histograms, that extends them adding auxiliary information like the full path-name of the histogram, its reference histogram (if present), and quality reports.
- the *quality tests* package, which is responsible of configuring and performing quality checks on specific ME, saving the outcome in properly formatted flags that are used while
performing the final data certification

- the DQMService, which is the software layer that links the DQMStore directly into CMSSW
- the DQMNet, which is the network layer that is responsible of connecting many DQM applications, from many machines, to a centralized web application, the DQMGUI, that is responsible for collecting and aggregating the received data.

3. Improvements in view of RunII

The DQM Framework has been a central component of the CMS experiment, with extended and deep connections both with the Online data processing and, also, with the Offline data reconstruction and certification. If, on one side, this implies that the DQM has been designed with flexibility in mind, on the other it also means that, any time changes and improvements are introduced in the surrounding systems, the DQM has to take them into account and adapt consequently. That is precisely what have happened to both Online/Data acquisition (DAQ) and to the CMSSW reconstruction frameworks during RunI. The changes that have been introduced, their challenges for the DQM system and the solutions that have been adopted will be explained in the following sections.

3.1. Online DQM Changes

A pictorial view of the Online DQM framework as it was during RunI is illustrated in Figure 3(a). Data flowed from the detectors directly into the high level trigger (HLT) farm, that selected events interesting for physics. The stream of events coming out of HLT was sent in its entirety to Tier-0 to perform the prompt reconstruction. A small, configurable, fraction of them, roughly 10%, was sent to the so called Storage Manager Proxy Server (SMPS), that represented the connection between the DAQ realm (depicted in the upper part of the figure) and the DQM one (depicted the bottom part of the figure). The SMPS was developed and maintained by the DAQ team in CMS and had many interesting characteristics that were extensively used during RunI:
• it was completely *network* based: this guaranteed a complete decoupling between the HLT processing pace and the rate with which the DQM Applications could analyze data. Moreover, the SMPS always served the most up-to-data available data to each request coming from the downstream DQM Application, assuring an ideal *live* monitoring.

• the SMPS was able to accept also selection criteria from the downstream DQM Applications, so that only interesting events were served to specific applications. This helped reducing the rate of exchange of events, while maximizing the efficiency of the DQM Applications.

• the SMPS had also the duty of collecting data quality monitoring information produced directly in the HLT farm, in the form of ROOT histograms, and merge it across the full HLT farm and serve it to dedicated DQM Applications. Unfortunately, the time required to perform the full merging of this information was too high to keep the pace of data taking, and a prescaling had to be introduced.

During the Long Shutdown I (LS1), the DAQ team re-designed and improved the system: unfortunately the SMPS disappeared in the newly implemented system. A schematic view of how the new system will look like during RunII is illustrate in Figure 3(b). The key changes that have been introduced in the new systems are:

• the handling of the data coming out of the HLT farm is no more based on a continuous stream of events, but it is fully file based: each produced file will contain the full statistics of a very limited period of data acquisition, which correspond to $\sim 23s$ (a so called *lumisection* (LS))

• the communication between the DAQ and the DQM realms is no more based on a *network protocol*, but is completely handled via a shared network file-system. This has the main advantage of being extremely stable, tested and does not need to be developed and maintained inside the CMS experiment.
the merging of the monitoring information produced in the HLT farm is not anymore a responsibility of the DAQ systems: the output files containing the histograms produced will be handled directly to the DQM team, whose responsibility is to aggregate them and expose their content via the DQMGUI.

The new design achieved a much better separation of responsibilities between the DAQ and DQM team, giving back to DQM what should be DQM’s. This, though, required a deep redesign of the processing logic of the online DQM data. First of all, the live nature of the online monitoring had to be preserved. In order to do that, a new mechanism to read and analyze data had to be introduced in the CMSSW framework. Its main characteristics are:

- an automatic file discovery tool.
- Automatic start&stop of DQM Applications.
- Embedded event selection.
- Dedicated histogram merging utility.

3.1.1. Automatic file discovery tool Every time a new file is produced and released by the HLT farm, the DQM Applications are automatically notified and have the freedom to perform the most appropriate action, depending on the desired output: either close the currently open file and immediately skip to the most recent one, in order to be as live as possible; or simply queue the new file and keep on processing the currently open file, in order to accumulate more statistics. To improve the flexibility of the system, the different behaviors can be easily tuned and modified on-the-fly via configuration files.

3.1.2. Automatic start&stop of DQM Applications During RunI the DQM Applications were in direct control by the central DAQ middle-ware (XDAQ), which proved to be somewhat complex, since it required the mixing of two completely heterogeneous frameworks, i.e. CMSSW and XDAQ. In the new systems, the DQM Applications are thought of as Software as a Service (SaaS) and they are always alive, waiting for file to appear on specific directories to start immediately processing data. The communication channel with the central DAQ has not been completely dropped. The applications are still sending heart-beat information to tell that they are working properly.

3.1.3. Embedded event selection The selection of particular events for each specific application has now been included in the automatic file discovery tool.

3.1.4. Histogram merging utility A new, dedicated utility, fastHadd, has been developed and deployed by the DQM team in order to be able to aggregate all the monitoring information produced in the HLT farm in order to be able to keep the pace of HLT event processing. This work required the introduction of a new file format which is based on Protocol Buffer[6], which internally still uses the native ROOT objects, serialization and summing routines. The careful handcrafting of the new file-layout allowed to significantly boost the performances, as illustrated in Table 1. The performances have been computed summing various numbers of files whose content was similar to the one produced by the HLT farm.
Table 1. fastHadd performances compared to native ROOT’s hadd.

| Merging Utility | 10 files | 30 files | 50 files |
|-----------------|----------|----------|----------|
| ROOT’s hadd     | 10.8s    | 48.9s    | 125s     |
| fastHadd        | 3.8s     | 10.2s    | 17s      |

3.2. Offline DQM Changes

The DQM framework is not a stand-alone software but it’s fully integrated into the more generic reconstruction framework of CMS, namely CMSSW. During the LS1, the requirements of fully exploiting the available computing resources while processing data, caused a profound redesign of the CMSSW framework. The need of reducing the memory footprint of a single job and the willingness of profiting of the multi-core processors available nowadays, convinced the CMSSW major architects to shift to a multi-thread paradigm. The multi-thread CMSSW is capable of processing many events in different threads in a single process. The changes needed to achieve such a result have been many and at the heart of the framework itself, hence deeply affecting also DQM. In particular, the two main challenges that the DQM framework had to solve in order to become multi-thread friendly and comply with the new policy have been represented by:

- the DQMStore. As explained in the previous sections, the DQMStore is acting as a unique, shared memory block that is valid across the full lifetime of the job: this causes many problems in a multi-thread environment, since concurrent writing access to the same (shared) memory location could results in data races or, even worst, unpredictable results.

- the ROOT histograms. The central tool used to perform data quality monitoring in CMS is represented by ROOT histograms. These are held in memory into the DQMStore and, unfortunately, are not thread-safe objects: this means that no concurrent writing access to them can be granted to users, which goes in the opposite direction with respect to the current implementation of the DQM framework, in which the ME holding the ROOT histograms are given back to users, exposing the code to all possible nasty conditions.

Two solutions have been explored in order to make the DQM framework thread-friendly. The first one involved the implementation of thread-safe histograms: this would have avoided at the root all possible data-race conditions, since the handled objects would have been intrinsically thread-safe. Unfortunately this solutions was not possible both because the ROOT team did not embrace this effort and, most importantly, because it would have not completely solved all problems related to multi-threading and running on several runs in a single job. The second solution was based on the multiply&serialize motto. In order to avoid all data-races related to histogram handling, these have been made local to each thread. The only remaining bottleneck was represented by the simultaneous booking of the same histogram from different threads: to solve also this problem, the writing access into the DQMStore’s memory has been guaranteed to be sequentially executed using a lock and a properly developed booking interface. This interface completely hides the locking handling from the end users who, in fact, will not event notice its existence, preventing them from the most common mistakes. A new aggregation step has been introduced in order to fetch the different copies of the same histograms from the different threads and sum them up: to perform this action we simply used ad-hoc hooks that have been natively designed into the multi-thread CMSSW framework exactly for this purpose. The simplified logic of histogram booking, summing and saving is illustrated in Figure 4. The changes required to accomplish the transition to the multi thread-friendly DQM have been huge: more that four hundred classes, together with their helper functions, have been migrated. It took more
than twelve months. The migration is mostly over (∼90%) and, so far, no major design flows appeared. Moreover, several validation campaign have been performed to compare the results obtained running the DQM code in single thread and multi-thread mode: the results have been identical, as expected. Preliminary performance measurement show that the multithread DQM framework is somehow penalizing the overall memory gain achieved thanks to the adoption of multithreading in the full CMSSW, while still saving CPU time.

![Diagram](image.png)

**Figure 4.** Booking, filling, aggregation and saving of histograms in the multi-thread friendly DQM framework.

### 4. Outlook and conclusions

The DQM framework performed extremely well during RunI and proved to be flexible and stable. During the Long Shutdown I the DQM framework successfully adapted and improved in the face of fundamental changes in the online and offline environments.

New functionality have also been added. For example, the possibility to compare data and Monte Carlo(MC) samples directly in the DQM Framework and show the results directly in the DQMGUI have been implemented, together with the capability to stack and display, again in the DQMGUI, more than one MC sample against data. The goal is to validate the simulation of the detector against the data in a central way and on a regular basis, including this step into the bi-weekly release validation campaign. Finally the review of the content of the histograms and the optimization and tuning of the quality criteria and the alarms has been carried out in order to have a fully automated certification of the quality of the acquired data.
References
[1] Chatrchyan S et al. (CMS) 2008 JINST 3 S08004
[2] Evans L and Bryant P 2008 JINST 3 S08001
[3] Tuura L, Eulisse G and Meyer A 2010 J.Phys.Conf.Ser. 219 072055
[4] Cerminara G (CMS) 2010 PoS ICHEP2010 485
[5] ROOT A data analysis framework http://root.cern.ch
[6] Google Protocol Buffer https://developers.google.com/protocol-buffers/