DQM4hep

A generic data quality monitoring framework for HEP

Rémi Ete[1,*] and *Antoine* Pingault[2,**]

[1]DESY, Notkestraße 85, 22607 Hamburg, Germany
[2]Ghent University, Department of Physics and Astronomy Proeftuinstraat 86, B-9000 Gent, Belgium

Abstract. Data quality monitoring is the first step in the certification of data recorded for offline physics analyses. Many experiments have developed their own dedicated monitoring system in the past. Most of them rely on their own event data model, which leads to a strong dependency on the data format and storage. We present here a generic data quality monitoring system, DQM4hep, that has been developed without any assumptions on the underlying event data model. This reduces the code maintenance and increases the portability and reusability across other experiments. We first introduce the framework architecture and the various core components as well as tools provided by the software package. We then give an overview of the different experiments using DQM4hep and the foreseen integration in future other experiments. We finally present the ongoing and future software development for DQM4Hhep and long-term prospects.

1 Introduction

Online data quality monitoring systems are crucial while taking experimental data. The main tasks of such tools are to provide a quick overview of the detector and sub-detector status and to evaluate the quality of their data in real-time. More technically, the online framework should provide the following ingredients:

- a link to the data acquisition system (DAQ) to access the detector data and run control status.
- a dedicated online analysis and data quality assessment framework.
- a visualization system to show the resulting analysis products.

Such systems are generally (but not always) decoupled from the data quality monitoring done offline on recorded data or simulated data. Often, the architecture of these software relies on the event data model of the underlying experiment. This can be seen as a limitation, as the software becomes non-reusable by other experiments.

In this paper, we present a new generic framework for online and offline data quality monitoring called DQM4hep. After introducing the framework architecture and specifications, the different components of DQM4hep are described. The software still being in development, we conclude with a few perspective and incoming features.

*e-mail: remi.ete@desy.de
**e-mail: antoine.pingault@ugent.be

2 The DQM4hep framework

2.1 Software architecture and specifications

The DQM4hep framework is built on top of the so-called plugin system component. A plugin is a C++ class encapsulating a user class that is compiled in a shared library and loaded at runtime by the plugin system. The plugin is used as a factory to create instances of the user class on demand. Such a system allows for building a framework with which the behavior of the key components can be changed at runtime.

In order to have a really generic monitoring system, the software should not depend on any event data model or data format. This allows having highly-reusable software for different experiments. By construction, in DQM4hep, the data type is user-defined and wrapped in a higher level handler (`dqm4hep::Event`). As the data has to be transported over the network, the data streaming, file reading, and writing facilities have to be user-defined. The user-streaming interface can be encapsulated in a plugin that the framework can use in online applications to serialize (de-serialize) data before sending (after receiving) to (from) an endpoint.

An important requirement for the online component of the monitoring system is to be distributed. A typical user analysis handles tens or hundreds of histograms that are filled whenever data are received from the DAQ system. The analysis program, in this case, is particularly *greedy* in terms of memory. Analyzing data in a single process is thus very limited and must be distributed over different processes or computers depending on the available resources on each machine. Since the resources are shared, this also increases the speedup of analysis and allows for more data to be processed in the same amount of time. The online analyses applications still have to be carefully designed in terms of memory management as they may run for many hours and must stay stable.

Before analyzing them, data needs to be forwarded from the DAQ system to the monitoring system. This mechanism may be of many forms: shared-memory (if running on same computers), network (TCP/IP), database, etc. but a major requirement is to avoid slowing or crashing the acquisition process.

2.2 The core components

The plugin system

As stated before, DQM4hep is built on top of a plugin system. The `dqm4hep::PluginManager` class is a singleton class holding all plugins within an application. The plugins are loaded by opening shared libraries using the `::dlopen()` function at startup. A user class object can be created on the heap by querying a specific plugin to the `dqm4hep::PluginManager` and by calling `dqm4hep::Plugin::create()`.

The event data model

The core component of DQM4hep defines a high-level handler, `dqm4hep::Event`, wrapping a user-defined event class. This handler is particularly useful because it allows transporting the user-defined event through an application workflow without making any assumption on the underlying implementation:

```
1  // a user-defined function
2  dqm4hep::EventPtr createEvent() {
3    // a DQM4hep event ...
4    dqm4hep::EventPtr event = dqm4hep::Event::create<MyEvent>();
```

```
5     // ... wrapping a user-defined structure
6     MyEvent *myevt = event->get<MyEvent>();
7     myevt->setTimeStamp(time(0));
8     myevt->setData({0.5, 856., 485.});
9     return event;
10  }
```

Event streaming

To send (receive) events to (from) the network, events have to be converted from or to a format that the transport system can handle (e.g binary). As the event is user-defined, the event streaming also has to be user-defined. The framework defines a base class `dqm4hep::EventStreamerPlugin` to stream in and out events that users have to implement:

```
11  class EventStreamerPlugin {
12  public:
13    // creates user-defined event
14    virtual EventPtr createEvent() const = 0;
15    // write event to TBuffer object
16    virtual StatusCode write(EventPtr event, TBuffer &buffer) = 0;
17    // read event from TBuffer object
18    virtual StatusCode read(EventPtr event, TBuffer &buffer) = 0;
19  };
```

The streamer implementation can be declared as a plugin and loaded at runtime using the `dqm4hep::PluginManager`. The combination of `dqm4hep::PluginManager`, `dqm4hep::Event` and `dqm4hep::EventStreamerPlugin` gives the fully required flexibility for such a framework.

Monitor elements

The concept of *monitor element* is central in data quality monitoring software. It encapsulates a summary of the data being monitored by the framework. One dimension histograms are the most commonly used monitor element types in HEP, but it can also be scalar values (counters), time-stamped graphs (environmental variables), 2D histograms (hit maps), etc.

Quality test and report

Whereas the monitor element provides a summary of a data set, the *quality test* implements the logic of testing an element or evaluating its quality. These tests are also designed to detect problems as some observables (mean of distributions, counters, etc.) may deviate from expectations during or after data taking. Figure 1 shows the workflow of monitor elements testing in offline mode.

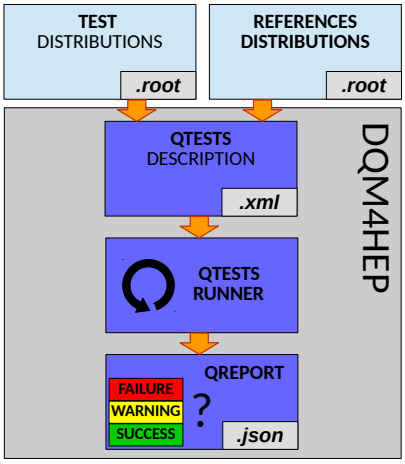

Figure 1. Offline quality test processing workflow: from monitor elements and references to quality test reports.

Monitor elements are stored as ROOT [1] objects with potential reference entities provided by experts. Monitor elements can thus be compared to the references as part of a quality test

or at a later stage by an operator. The quality tests and the monitor elements to test are described in XML format and passed to a program for processing. A *quality test report*, containing quality test status (success, warning, failure), quality estimate (in the range [0;1]) and additional custom user data, is output in JSON format.

2.3 The online system

The main purpose of a data quality monitoring system is the visualization and quality assessment of detectors data in a real-time environment.

Collectors

The first stage in achieving this objective is the transmission of the data from the DAQ system to the analyses modules. This is done by building and wrapping user-defined event from the DAQ data as DQM4hep events before streaming them to the analyses modules. Since there might be an indefinite number of modules running on multiple hosts, there is a need for a central access and distribution point of this data. This is the role of the collectors, of which two types are defined:

- The *event collector*, which collects `dqm4hep::Event` from the event builder and distributes them to the analyses modules.
- The *monitor element collector*, which collects the products of these analyses, the *monitor elements*, and distributes them to the visualization interfaces.

Modules

Two major data flavors are needed when running an experiment, the physics data from the detector itself and the data from the slow control systems. To cope with these two possible kinds of input, two types of modules are defined within the framework: the analysis module and the standalone module. They are both user-defined plugins that can be started or stopped at any point while the framework is running. Their goal is to reduce the initial amount of data to a few useful elements, the *monitor elements*. Quality tests can assess the output data and resulting quality test reports stored within the *monitor element* and sent along to the *monitor element collectors*.

Where analyses modules use DQM4hep event streamed by the `event collectors` as input, the standalone modules use data from external sources (pressure sensors, detector high voltages, etc.). These external sources, and thus the modules, operate independently of the DAQ system. A standalone module can include some analysis but is most of the time just used to process the raw external data for visualization and data quality assessment. Analyses modules can be run independently in online or offline mode. In this context, offline means there is no detector system running, and no operator is waiting for the data to be displayed continuously. There is thus no incentive to process and display the data in real time: the data is read and stored from and to disk, and no run management is needed. This mode is especially useful for quality assessment of simulation data and new analyses testing.

A visual representation of the DQM4hep online architecture is shown in Figure 2.

2.4 The online visualization

To ease the use of the framework, several Graphical User Interfaces (GUI) were implemented. For now, only QT [2] based interfaces are fully functional and available for end-users.

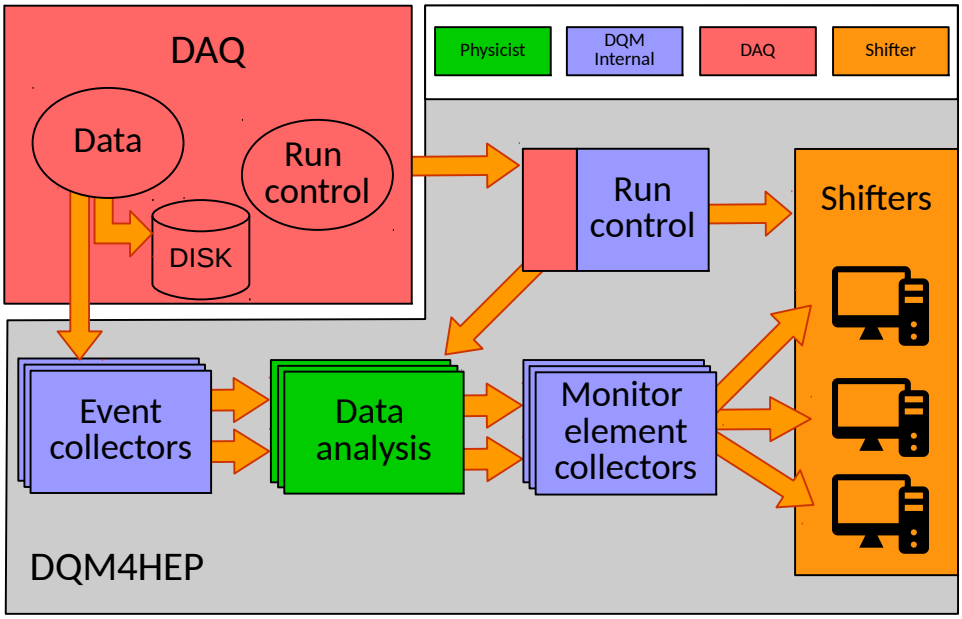

Figure 2. Online architecture of the DQM4hep framework. Data flow is indicated by the arrows. Physicists only have to dedicate time on the data analyses. The top left corner corresponds to the DAQ system and is not part of the framework, it is displayed here to show the link between the two systems.

Job control

The framework needs multiple applications (jobs) to run at the same time (collectors, modules, run control server, etc.), sometimes over different host for load balancing purposes. For practical reasons, it is important to easily control and monitor these processes from a central point. This is achieved through the use of *Job control servers* that run as daemons on every host taking part in the deployment. They wait for remote client interfaces, called *Job control interfaces*, to connect and manage the local processes. A GUI, the *job control interface*, is used as a central point to monitor and control all the jobs across all the hosts. When using this interface, all the jobs needing to run have to be defined and configured (process name, host, options, etc.) through a JSON configuration file. To simplify this configuration process, the framework implements a JSON parser which extends the default JSON functionalities such as comments and variable expansion.

Main Monitoring Window

Most users will predominantly use only this interface as its function is to display the end results of the analyses and quickly give an overview of the quality of the data being taken. A general view of this interface can be seen in Figure 3. The first step in using this interface consists to browse the available `Monitor Element Collectors` and select the `Monitor Elements` of interests. This is done through the *browse* window which provides various search function (by ME name, module, etc.) to help construct this selection. A tree-like structure, with folders and sub-folders, is then available on the left side of the main interface to navigate previously selected ME. Apart from the top-level folders, which are named after the analysis module the ME belongs to, the organization of this structure is user-defined

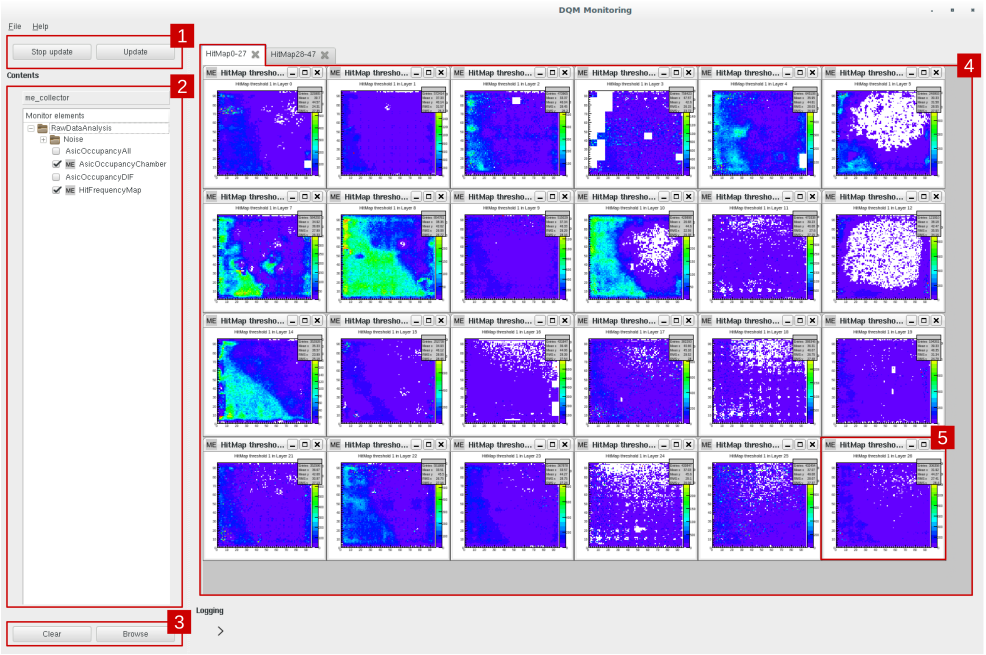

Figure 3. Main window of the monitoring GUI. 1: Option for manual/auto update of the visualization. 2: Selected `Monitor Elements` (`ME`) organized in a tree-like structure 3: Open a *Browse* window to create the `ME` selection available in 2. 4: Drawing section for `MEs`, it can be organized in multiple tabs. 5: Drawn `ME` are valid ROOT object that can be manipulated (zoom, scale change, fit, etc.)

within each module. The user can then select which `ME` to display in the drawing area. Once displayed, they can be rearranged at will within the drawing area and multiple tabs can be opened to show different subsets of interesting `ME`. The icon background of each `ME` is colored depending on the results of its selected quality test for a quick visual assessment of its quality from the user. Multiple tests can be assigned to each `ME` and their details and report can be accessed via a context menu. As mentioned before each `ME` is a valid ROOT object and can, therefore, use all the functionalities ROOT offers (fitting, re-scaling, etc.). This proves particularly useful during data taking to quickly glance and analyze a specific subset of data. Multiple instances of this interface can be opened on any host, provided they are on the same network and connect to the available *Monitor Element Collectors*.

Web Interface

A web-based interface regrouping all the GUI mentioned has also been developed. As of this writing, it is still missing some key links to the back-end to be fully operational for the end-users. The main advantage of such interface is the portability as the only prerequisite is a recent internet browser.

3 Detectors using DQM4hep

The work on this framework was started within the CALICE [1]-Semi Digital Hadronic CALorimeter (SDHCAL) collaboration. As such, a dedicated implementation, based on

[1] CAlorimeter for LInear Collider Experiment

the LCIO [3] Event Data Model and including multiple analyses modules, was produced to demonstrate the capabilities of the framework. For the past three years and five test beam campaigns, the SDHCAL prototype successfully used this tool as their online data quality monitoring (DQM) system. Since then, other experiments from the CALICE collaboration adopted the framework for their DQM needs. It has also been deployed for a combination of two separate detectors using different data acquisition systems, demonstrating the reusability and scalability of the system.

New experiments from other collaborations such as the DREAM calorimeter from RD52 have shown their interest and started integrating this solution. Work on coupling with the generic data acquisition framework EUDAQ [4] have also started. Such integration would provide powerful, almost off-the-shelf, sets of tools for experiments to acquire and assess the quality of their data.

4 Conclusion

A new generic framework for data quality monitoring for high energy physics, DQM4hep, has been developed without any assumption on the underlying event data model implementation or data format. These core guidelines promote high degrees of portability and reusability across experiments. This software solution provides tools to develop data analyses and quality assessment of any experiments data. It can be deployed for both online (i.e real-time data taking) and offline (i.e simulation or pre-recorded data) setups.

Visualization interfaces have been developed with Qt4 libraries for steering the online deployment and displaying the monitor elements collected during data taking. The framework has now been used in real-world condition by several detector prototypes across multiple test beam campaigns for their monitoring needs, showing the reusability of the software. For some of these campaigns, the data acquisition was performed with multiple sub-detectors, and the framework deployed across several computers, demonstrating the scalability of the software.

Web tools are currently in development to replace the GUI tools and improve the portability of user interfaces.

References

[1] R. Brun, F. Rademakers, Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment (1997)

[2] The Qt Company, *Qt 4.8* (2011), https://www.qt.io/

[3] S. Aplin, J. Engels, F. Gaede, N.A. Graf, T. Johnson, J. McCormick, *LCIO: A persistency framework and event data model for HEP*, in *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)* (2012), pp. 2075–2079, ISSN 1082-3654

[4] EUDAQ Software Developers, *A generic data acquisition framework* (2018), http://eudaq.github.io