Data quality monitoring tools ready

Ete, R. (DESY) et al

06 November 2017

The AIDA-2020 Advanced European Infrastructures for Detectors at Accelerators project has received funding from the European Union’s Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.

This work is part of AIDA-2020 Work Package 5: Data acquisition system for beam tests.

The electronic version of this AIDA-2020 Publication is available via the AIDA-2020 web site or on the CERN Document Server at the following URL:

Copyright © CERN for the benefit of the AIDA-2020 Consortium
Abstract:

This document summarizes the development of data quality monitoring tools and their use by different detectors during test beams. These tools are part of a software package called DQM4HEP [1]. Its development has been steered with 'genericity' and 'extensibility' in mind and therefore it provides all necessary tools to monitor data for any detector in any format. The various online tools developed for two different detectors, the SDHCAL [2] and AHCAL [3] prototypes are presented as proof of principle of the DQM framework. Further development is foreseen to link this software to directly get data from the EUDAQ [4] data acquisition framework but also to extend it to monitor data in offline mode.
AIDA-2020 Consortium, 2017
For more information on AIDA-2020, its partners and contributors please see www.cern.ch/AIDA2020

The Advanced European Infrastructures for Detectors at Accelerators (AIDA-2020) project has received funding from the European Union’s Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168. AIDA-2020 began in May 2015 and will run for 4 years.

Delivery Slip

<table>
<thead>
<tr>
<th>Name</th>
<th>Partner</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authored by</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R. Ete</td>
<td>DESY</td>
<td>20/10/2017</td>
</tr>
<tr>
<td>T. Coates</td>
<td>USSUS</td>
<td></td>
</tr>
<tr>
<td>A. Pingault</td>
<td>UGENT</td>
<td></td>
</tr>
<tr>
<td><strong>Edited by</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L. Lapadatescu</td>
<td>CERN</td>
<td>20/10/2017</td>
</tr>
<tr>
<td><strong>Reviewed by</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M. Wing [WP5 coordinator]</td>
<td>UCL</td>
<td>06/11/2017</td>
</tr>
<tr>
<td>F. Sefkow[Scientific coordinator]</td>
<td>DESY</td>
<td></td>
</tr>
<tr>
<td><strong>Approved by</strong></td>
<td>Scientific Coordinator</td>
<td>06/11/2017</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

1. Introduction.......................................................................................................................................................... 5  
2. The data quality monitoring concept................................................................................................................. 6  
3. DQM4hep as a generic framework ....................................................................................................................... 7  
   3.1. The CORE concepts......................................................................................................................................... 7  
   3.2. The central objects of the framework............................................................................................................ 8  
   3.3. The online architecture.................................................................................................................................. 8  
4. Use of DQM4HEP in various BEAM testS ........................................................................................................ 14  
5. Collaboration page and release ............................................................................................................................ 16  
6. Summary and Outlook ......................................................................................................................................... 17  
7. References........................................................................................................................................................... 18  
8. Annex: Glossary...................................................................................................................................................... 19
Executive summary

In this document, we report the development of data quality monitoring tools and their use in various test beam situations, with different detector technologies and for combinations of such detectors. The software described in this document refers to the version v01-04-04 (see section 5).

Section 1 introduces the development of monitoring tools in the context of AIDA-2020.

Section 2 describes the concept and challenges to ensure the monitoring of data quality in a test beam configuration or a full-size experiment.

Section 3 describes in detail DQM4HEP, the software solution developed to achieve this task.

Section 4 shows the use of DQM4HEP in various beam tests involving different detectors and combination of detectors of the CALICE collaboration and other device types.

Section 5 provides url links to releases and support pages.

Section 6 summarizes this report and discusses the future work and extensions of the framework.
1. INTRODUCTION

Data quality monitoring is the first step to the certification of the recorded data for off-line physics analysis. Dedicated monitoring frameworks have been developed by many experiments in the last decades and usually rely on the event data model (EDM) of the experiment, leading to a strong dependency on the data format and storage. We present a generic data quality monitoring system that has been developed without any assumption on the EDM. This increases the code maintainability and the portability across different experiments.

In this document, we present the functionalities of this framework, DQM4HEP [1], and its philosophy. We describe where the different users (developer, shifter, engineer) need to focus to make the framework usable for a test beam setup or a full-size detector such as the ILD detector of the ILC project. As a proof of principle, the framework has been deployed for many test beams, involving different detectors or combinations of detectors. After being successfully tested, further development is foreseen to directly get the data from the EUDAQ [4] data acquisition system and thus, make a generic and robust set of software for data taking.
2. THE DATA QUALITY MONITORING CONCEPT

It is important that data can be both examined and checked for its quality in real-time during a test beam. This is useful both for the understanding of hardware and for shifters who are taking data. Anomalous information can indicate such things as faulty wiring, unresponsive or dead channels, damaged detector components, and many other hardware faults or irregularities that are important to identify. While shifters are operating, online monitoring allows them to identify issues such as incorrect beam energy or placement, or reduced beam rates, both of which can affect the rate of data taking or the quality or usefulness of the information. This makes online monitoring extremely valuable to test beam operations.

Data quality monitoring assesses the accuracy of data being received in real-time. This allows test beam operators to identify whether their device under test is operating as expected and fulfilling their requirements for resolution, timing, etc. Data quality monitoring (DQM) uses a variety of methods to evaluate the quality or "goodness" of data received, often using statistical measures such as the mean, median and standard deviation, or comparison to references.

DQM4HEP is designed to be able to fulfil these requirements in a generic way. The structure of the program (discussed in more detail in Section 3) allows for independent components of the framework to be used, not used, or exchanged, by isolating the sections of the program that are specific to the user or use-case. The components that are case-specific – the analysis and standalone modules – are written in standard C++ code, meaning they are capable of performing any data unpacking, processing or analysis that is necessary; the framework then handles packaging this information in a useful way and networking to transmit it to where it is needed.

The flexibility of the framework is key to it being generic. The event data model (EDM) abstraction means that the framework does not need to have special rules for handling particular types or structures of events, which is what allows it to handle anything that can be packed into, decoded from, and accessed by normal C++ methods. The ability to run multiple instances of each process of the framework is also key to its flexibility. This allows users to, for example, run different analyses for different detector sub-components, operate in online or offline modes, and run multiple analyses in parallel or distributed over several networked computers to reduce computational load. These features are all useful for online DQM. Online data quality monitoring is a critical component of test beam software, necessary for real-time response to conditions in the test beam and helpful for further developing the software and understanding the response of the hardware. Offline mode allows data to be "re-run" through monitoring and DQM tests to compare discrepancies that may not have been identified during online run, as well as to iterate and test new or improved quality tests on old data to compare their results.
3. DQM4HEP AS A GENERIC FRAMEWORK

3.1. THE CORE CONCEPTS

The DQM4HEP software relies on two principles: modularity and genericity. The core part of the framework is based on a plugin system that allows shared libraries to be loaded (so/dylib/dll) and hook classes at runtime for further use. The plugin classes are compiled in a separate shared library by the user. These classes must be registered by using the macro `DQM_PLUGIN_DECL()` as shown here:

```cpp
DQM_PLUGIN_DECL(UserClass, "UserClass");
```

The user class does not have to inherit from any framework class, making the mechanism easier to implement for users. The `dqm4hep::DQMPluginManager` class is then responsible for:

- loading the user shared libraries
- storing the plugins found in these libraries
- providing an interface to create a new instance of the plugin classes

Most of the programs provided by the different packages start by loading shared libraries before executing any other operation. The library names have to be specified through the environment variable `DQM4HEP_PLUGIN_DLL` before starting a program:

```sh
$ export DQM4HEP_PLUGIN_DLL=./lib/libMyUserPlugin.so:./lib/libMyUserPlugin2.so
# then start a program …
```

This mechanism allows us to provide a high modularity in the framework and gives the possibility to externalize highly specific implementations for the different experiments. The EDM and the event streaming implementations have been chosen to be a plugin. Users have to specify what kind of data they want to use and how to read and write them in a raw buffer. This behavior is encapsulated in a plugin class that must implement the `dqm4hep::DQMEventStreamer` interface with the following virtual methods:

```cpp
virtual dqm4hep::DQMEvent* createEvent() const = 0;
virtual StatusCode read(dqm4hep::DQMEvent*& event, xdrstream::IODevice* device) = 0;
virtual StatusCode write(const dqm4hep::DQMEvent* const event, xdrstream::IODevice* device) = 0;
```

where the `xdrstream::IODevice` generally handles a raw buffer in memory and provides an API to read and write raw data. This class belongs to the `xdrstream` package, dedicated to stream data with different models (raw buffer, file, socket). The `dqm4hep::DQMEvent` holds the real user event model in memory and can be accessed by using the `DQMEvent::getEvent<T>()` method, with the
real user event class. This approach leads to a framework that is completely transparent regardless of the real event implementation.

### 3.2. THE CENTRAL OBJECTS OF THE FRAMEWORK

The goal of this framework is to assess the data quality while taking data. Two classes play a central role in the software:

- the `dqm4hep::DQMMonitorElement` class, mainly holding a ROOT::TObject object from the ROOT framework [5]. This class represent the object to monitor within the framework. The list of the class members is shown in table 1.
- the `dqm4hep::DQMQualityTest` class, implementing the logic of how to test a `dqm4hep::DQMMonitorElement`. The quality test result provides an estimate of the 'goodness' of the tested object.

Here is a simple example a calorimeter prototype is exposed to a 20 GeV pion beam, and the total energy distribution during the run is plotted. One wants to know if the total measured energy in your data is as expected within a certain error. In the DQM4HEP framework, the `dqm4hep::DQMMonitorElement` object will contain a one-dimensional histogram (ROOT::TH1F) and the quality test will calculate the mean of the distribution and check if it is contained in a certain range around 20 GeV. The closer the mean is to 20 GeV the better the quality will be. Note, that in this case, the mean is maybe not enough to estimate the quality of the distribution. Thus, multiple quality tests can be performed on a single element to test its 'goodness', leading to a more robust way to assess the quality of the data.

### 3.3. THE ONLINE ARCHITECTURE

The global picture of the online architecture is shown in figure 1.

Each box represents a separate process which are linked by network communication, either TCP/IP or HTTP. The TCP/IP communication is handled by DIM [6], a light software developed by the DELPHI experiment, and the HTTP communications by Mongoose [7]. The color code refers to the different users that have to interact within the framework while deploying it for a given setup.

The DAQ box on the left is tagged as an engineer task as the link between the DAQ and the DQM4HEP framework relies on the EDM and event streaming. This is usually implemented once by the DAQ engineer who is usually more familiar with low level programming.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TObject</td>
<td>m_pObject</td>
<td>The ROOT object to monitor</td>
</tr>
<tr>
<td>std::string</td>
<td>m_name</td>
<td>The monitor element name</td>
</tr>
<tr>
<td>std::string</td>
<td>m_title</td>
<td>The monitor element title</td>
</tr>
<tr>
<td>std::string</td>
<td>m_description</td>
<td>A short description of this monitor element</td>
</tr>
<tr>
<td>std::string</td>
<td>m_drawOption</td>
<td>The ROOT object draw option</td>
</tr>
<tr>
<td>Type</td>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>std::string</code></td>
<td><code>m_moduleName</code></td>
<td>The name of the module that has booked this monitor element</td>
</tr>
<tr>
<td><code>std::string</code></td>
<td><code>m_collectorName</code></td>
<td>The name of the monitor element that collected this monitor element</td>
</tr>
<tr>
<td><code>DQMPath</code></td>
<td><code>m_path</code></td>
<td>The path (string) to the directory where the monitor element is stored</td>
</tr>
<tr>
<td><code>DQMQuality</code></td>
<td><code>m_quality</code></td>
<td>The quality flag evaluated by quality tests</td>
</tr>
<tr>
<td><code>DQMMonitorElementType</code></td>
<td><code>m_type</code></td>
<td>The type of monitor element</td>
</tr>
<tr>
<td><code>DQMResetPolicy</code></td>
<td><code>m_resetPolicy</code></td>
<td>A flag specifying when the monitor element has to be reset during data taking</td>
</tr>
<tr>
<td><code>unsigned int</code></td>
<td><code>m_runNumber</code></td>
<td>The run number during data taking</td>
</tr>
<tr>
<td><code>DQMTestMap</code></td>
<td><code>m_qTestMap</code></td>
<td>The list of quality tests to process</td>
</tr>
<tr>
<td><code>DQMTestResultMap</code></td>
<td><code>m_qTestResultMap</code></td>
<td>The quality test results</td>
</tr>
</tbody>
</table>

Table 1: The main attributes of the `dqm4hep::DQMMonitorElement` class

The blue boxes are DQM internal processes. The run control server is a single process within a setup that receives the “start of run” and end of run signals from the DAQ and forwards them to listening DQM4HEP applications. The current protocol in use to receive the signals is HTTP (using a POST from DAQ). Other simpler software can provide this functionality and be easily embedded in a DAQ without having the DQM4HEP dependency.

The event collector servers are the first entry points in the framework. Multiple collectors can be run independently within a setup. For example, one could collect events coming from the DAQ event builder and others from the different sub-detectors events.

The monitor element collector servers are the last exit points of data before inspection by the shifters. They collect monitor elements coming from user analysis (see below) and redistribute them to shifters on visualization interfaces (Qt). Again, multiple monitor element collectors can be run within a deployment. The different analyses can thus send their monitor elements to different collectors, reducing the load in terms of memory, bandwidth and CPU for single collectors.
The orange boxes refer to so-called "DQM modules". These applications process data and produce monitor elements that are collected by monitor element collectors. As the setup is specific to an experiment, the analyses are also specific and thus have to be developed according to the data sent by the DAQ. As for the EDM, the analyses are encapsulated within plugins. They are loaded at runtime by a dedicated application. More generally speaking, these applications could be considered as consumers as they treat data but also as producers as they produce monitor elements. In this case, we can qualify these applications as reducer. Two types of module can be distinguished:

- **the analysis module** that receives and processes data from the DAQ system. The user must implement the `dqm4hep::DQMAnalysisModule` class with the virtual methods as shown in the code below. The `initModule()` is called when the application starts and is called once. The `readSettings()` function allows the analysis to be configured using user specific settings from an XML section of the application configuration file. This is also the place where the monitor elements are booked within the application. The `startOfRun()` function is called each time a new run is started. The user get the different parameters from the DAQ in the `dqm4hep::DQMRun` object passed as function argument. The equivalent function `endOfRun()` is also called each time the current run ends. The two functions `startOfCycle()` and `endOfCycle()` are related to one of the most important parts of the application: the cycle structure. Indeed, in terms of performance, one can easily understand that hundreds of histograms cannot be sent across the network each time an event coming from the DAQ is processed. This would lead for example to a bandwidth overload. Even more, it makes no

---

**Figure 1: The global online architecture of DQM4HEP.**
sense for the shifter to update the visualized plots every half a second. The solution would be to group these updates more spaced in time. This is the definition of a cycle: a period during which events are processed and at the end of which monitor elements are sent to the dedicated collector. Different types are implemented:

- event counter cycle: ends after N events processed
- timer cycle: ends after N seconds
- event size cycle: ends after N bytes of data have been processed

The boxes at the bottom of figure 1 (job control servers and job control GUI) are related to job control functionalities. Indeed, the picture shows that during a deployment, the number of involved processes to manage is important. Starting manually each process on different hosts can be a heavy and painful task. To efficiently manage all these programs, job control servers are run as UNIX daemons on each involved host of the setup and manage the processes (start, stop, get status, get log files, ...). All these servers are steered remotely by the shifters through a dedicated user interface (see below).

Finally, the green boxes are the endpoints of the system: the visualization interface. The main one is the monitoring interface, written as a desktop application using Qt4 [8] libraries. A screenshot example of this interface is shown on figure 2. The main functionalities are the following:
- drawing of multiple plots on a canvas, each embedded in different named tabs. This allows for better grouping of plots, easier comparisons and better understanding of the data taking
- checking of quality test reports performed on the monitor element in the different modules
- browsing of the monitor element collectors content, using filters to display subsets of monitor elements
- saving and restoring the monitoring interface contents: monitoring element names, drawn plots, opened canvases, etc... This functionality is particularly useful for shifters who want to prepare test beams in advance and restore the interface in one click during the test beam.

The second interface is the job control client interface, also implemented as a desktop application using Qt4 [8] libraries. Its role is to provide a full remote control on the DQM processes during data taking. Its implementation does not depend on DQM4HEP and can actually be used as a generic job control. Figure 3 shows a screenshot of the job control client interface during a test beam. The list of processes to manage is written in a JSON file which is loaded by the desktop application. The interface allows processes to be started and stopped individually or by group. Various informations such as the process name, PID and status are displayed. This information can be updated in real time (if the user chooses so) with a customizable timer. Note also that the processes can be stopped by sending a custom signal. This is particularly useful when a process wants to catch a particular signal and perform an operation before exiting. When a process is starting, its standard output is redirected to a log file that can be retrieved by this client application (Open LogFile button).

Figure 2: The main monitoring window for shifters.
Figure 3: The job control client interface.
4. USE OF DQM4HEP IN VARIOUS BEAM TESTS

DQM4HEP has already been used in a number of beam tests for multiple detector prototypes, including combined test beam experiments. So far these have been with the AHCAL+beam telescope, and SDHCAL+SiWECAL. More specific information on these can be found in the references [2][3].

During SDHCAL beam tests, DQM4HEP was interfaced with the data acquisition system using a shared memory (shm) feature, allowing it to access information online. In AHCAL beam tests, the framework was used in "nearly-online" mode; completed runs in LCIO format were accessed by the LCIO file service over network-attached storage as soon as an individual run file was finished. The file service can be run on files as they are being written but as events are loaded into memory, only events that were present in the file at that time were available for monitoring.

DQM4HEP was extremely useful as a tool to identify issues with the detectors during testing. For example, during AHCAL tests, where new scintillator tile layers were being tested for the first time, the hitmaps allowed quick and simple visual identification of any scintillator tiles whose electronics were noisy or dead. These were plainly visible as erroneous hits for noisy channels, or blank squares for dead channels. In SDHCAL tests, the current in the resistive plate chambers is expected to be near 0 while taking data. Thanks to the monitoring it was possible to prevent starting bad runs when the monitored current was too high, saving time and various resources. Figure 4 also shows some hit maps of many GRPCs of the SDHCAL recorded during a test beam. Some of the chambers show a lack of hits in their center (e.g. row 2, column 6). This is due to an overflow of incoming gas in the chamber, which leads to its inflation and thus explains this lack of hits. By looking at these maps, we were able to spot the problem, correct for this and restart a new run with a more stable detector.

The following page shows a variety of plots from various beam tests, demonstrating the use of the framework both as a monitoring tool and a slow control tool. Figs. 4 (top-left), 4 (top-right), 4 (bottom-left) and 4 (bottom-right) are from a combined SDHCAL+SIWECAL beam test in June 2016 at the CERN-SPS. Figure 4 (middle-left) is from an AHCAL test at the CERN-SPS beam in May 2017, and figure 4 (middle-right) is from a combined AHCAL and beam telescope test at DESY in February 2017.
Figure 4: Various plots from the monitoring interface recorded during AHCAL and SDHCAL beam tests. (top-left) Hit map of layer six of the SDHCAL with the beam off, showing noisy channels, (middle-left) A plot showing a hadronic shower within the SDHCAL, (top-right) Hit map of a single layer of the AHCAL, (middle-right) Spatial correlation plots of the AHCAL and beam telescope, (bottom-right) A plot of the ambient air pressure around the SDHCAL detector, (bottom-left) A plot of applied high voltage (HV) on each layer of the SDHCAL detector.
5. COLLABORATION PAGE AND RELEASE

Most of the information related to the DQM4HEP software can be found on the Github collaboration page:

https://github.com/DQM4HEP

The software version on which this document is based can be found on the release page:

https://github.com/DQM4HEP/dqm4hep/releases/tag/v01-04-04

Issues, questions and more general support from the DQM4HEP team can be found on this page:

https://github.com/DQM4HEP/dqm4hep/issues

A dedicated email has also been setup for user support:

dqm4hep@gmail.com
6. SUMMARY AND OUTLOOK

The DQM4HEP framework provides a reliable method for online monitoring and data quality monitoring for physics test beam data that is generic, flexible and scalable. It has proven that it can adapt to different detector types, including detectors with different event and read-out structures. It has also proven that it is capable of handling common test beams with more than one detector, correlating the information received from the data acquisition to create a picture of what is happening in all detectors.

Because of DQM4HEP's unique generic nature, it is ideally suited to a variety of test beam configurations. Although some additional work is required to achieve an online interface between DQM4HEP as a generic monitor and EUDAQ as a generic DAQ, the current version is still a capable and potent monitoring tool and fulfils the specifications for Milestone 67.

Work is also on-going to move all Graphical User Interfaces to a web browser-based UI. This will allow for central monitoring and management of the deployed systems without having to install the entire software package on shifters' computers. The second important step is to simplify the software for further analysis of online data quality checks. This will allow, for example, checking the data quality of different ILC Monte Carlo productions and quickly detect and solving problems before they propagate to physics analyses.
7. REFERENCES

https://doi.org/10.5281/zenodo.1012575


### 8. ANNEX: GLOSSARY

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAQ</td>
<td>Data acquisition system</td>
</tr>
<tr>
<td>ILC</td>
<td>International Linear Collider</td>
</tr>
<tr>
<td>ILD</td>
<td>International Large Detector</td>
</tr>
<tr>
<td>XDRSTREAM</td>
<td>eXternal Data Representation Streaming package</td>
</tr>
<tr>
<td>EUDAQ</td>
<td>A generic data acquisition framework</td>
</tr>
<tr>
<td>DQM</td>
<td>Data Quality Monitoring</td>
</tr>
<tr>
<td>DQM4HEP</td>
<td>Data Quality Monitoring for High Energy Physics</td>
</tr>
<tr>
<td>EDM</td>
<td>Event Data Model</td>
</tr>
<tr>
<td>ROOT</td>
<td>A data analysis framework for physicists</td>
</tr>
<tr>
<td>DIM</td>
<td>Distributed Information Management</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>Qt4</td>
<td>Library for desktop GUI implementation</td>
</tr>
<tr>
<td>Mongoose</td>
<td>A HTPP client and server library written in C/C++</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>CALICE</td>
<td>Calorimeter R&amp;D for Linear Collider detectors collaboration</td>
</tr>
<tr>
<td>SDHCAL</td>
<td>Semi Digital Hadronic CALorimeter</td>
</tr>
<tr>
<td>AHCAL</td>
<td>Analogue Hadronic CALorimeter</td>
</tr>
<tr>
<td>LCIO</td>
<td>Linear Collider Input Output persistency framework and event data model for linear collider detector studies</td>
</tr>
</tbody>
</table>