

PAPER • OPEN ACCESS

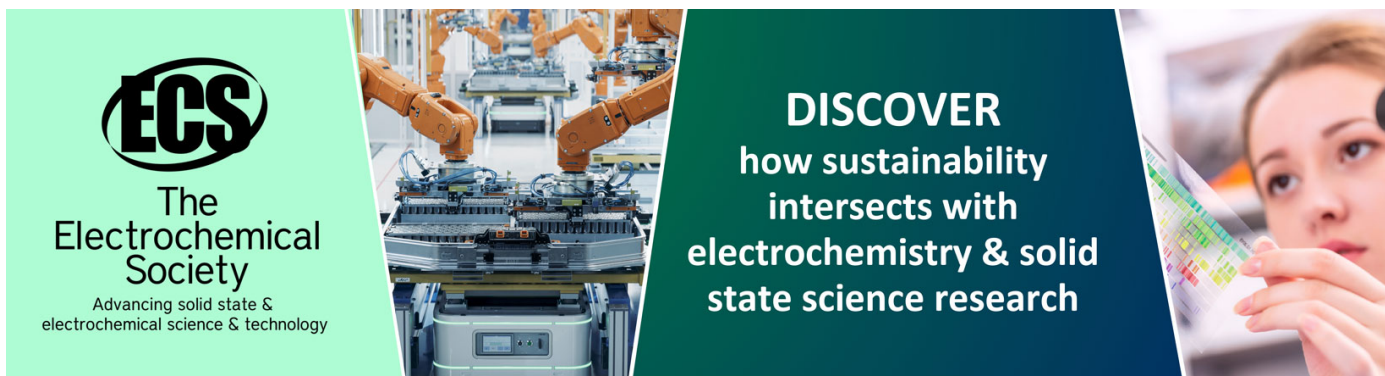
## The DQM system for the SND detector

To cite this article: K. V. Pugachev *et al* 2020 *J. Phys.: Conf. Ser.* **1525** 012069

View the [article online](#) for updates and enhancements.

### You may also like

- [Gaussian convolution decomposition for non-Gaussian shaped pulsed LiDAR waveform](#)  
Jinli Fang, Yuanqing Wang and Jinji Zheng
- [Photon-number-resolving superconducting nanowire detectors](#)  
Francesco Mattioli, Zili Zhou, Alessandro Gaggero et al.
- [An effective source number detection method for single-channel signals based on signal reconstruction and deep learning at low SNR](#)  
Yunwei Zhang, Zixuan Wei and Yong Gao



**ECS**  
The  
Electrochemical  
Society  
Advancing solid state &  
electrochemical science & technology

**DISCOVER**  
how sustainability  
intersects with  
electrochemistry & solid  
state science research

# The DQM system for the SND detector

**K. V. Pugachev<sup>1,2</sup>, L. V. Kardapoltsev<sup>1,2</sup> and A. A. Korol<sup>1,2</sup>**

<sup>1</sup> Budker Institute of Nuclear Physics, SB RAS, Novosibirsk, 630090, Russia

<sup>2</sup> Novosibirsk State University, Novosibirsk, 630090, Russia

E-mail: K.V.Pugachev@inp.nsk.su

**Abstract.** The SND detector has been operating at the VEPP-2000 collider (BINP, Russia) for several years. We present its new DQM system.

The system deals with multiple parameters being numbers, histograms or user opinions to form a data quality decision (good, bad, etc). It calculates SND subsystem and overall statuses and produces summaries available via web-interface. The system supports multiple parameter sets for different usages like daily checks, precise subsystem control, preparing data for processing, etc. The parameter values are analyzed using special ROOT macros yielding quality checks that could be supplemented or corrected by users.

The system is a part of a new SND information system implemented as a web application using Node.js and MySQL. The ROOT macros are written by SND subsystems experts using a simple framework. Histograms are displayed using JSROOT.

The system is put into production. It is consistent, however there are some improvements to implement.

## 1. Introduction

The SND detector [1, 2, 3] operates at the VEPP-2000 collider [4]. Its present version was introduced in 2008. The SND experiment produces hundreds of gigabytes stored data having upper theoretical limit of 25Gb/s [5] for raw data flow, megabytes of metadata per day. It is supported by a team of about 30 collaborators. To understand collected data quality thousands of histograms are generated. The histograms are associated with about twenty SND subsystems like calorimeter, drift chamber, muon system, DAQ, and others. Some system properties having physical meaning (like luminosity) are put in the SND offline database.

We decided to develop a data quality monitoring system that would provide next functionalities:

- show run state, parameters and histograms,
- assign parameter quality by software and/or by user,
- group parameters by subsystems, subsystems by run,
- provide multiple parameter sets for different usages (e.g. for operators/experts/debug/new conditions),
- select run quality information by various criteria (e.g. runs having bad data quality according to a specified subsystem/parameter or runs having special quality pattern for a set of parameters e.g.  $p_1$  is good,  $p_2$  is bad and  $p_3$  is in between).



Here and below a minimal set of events for analysis is referred to as “run”. The main goals of this are to have a single access point to all the quality data and user friendly interface for easy manipulating them.

## 2. Architecture

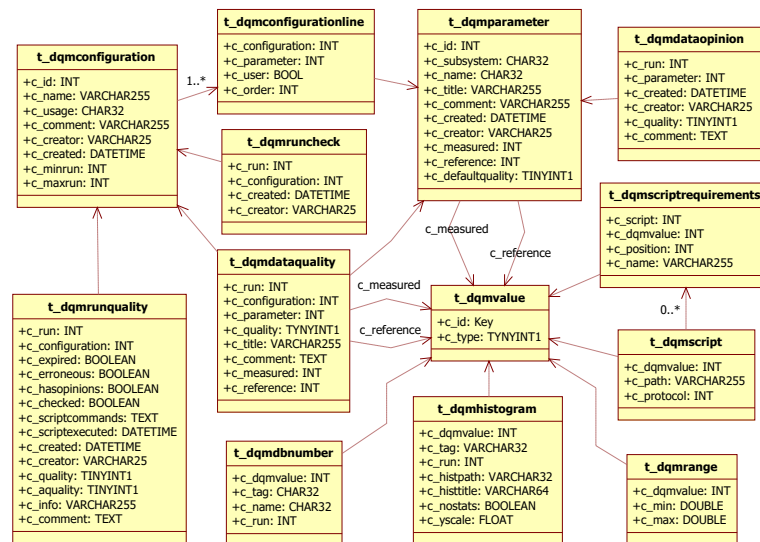


Figure 1. The DQM database structure.

The DQM is a part of the new SND information system implemented as a client-server Node.js [6] web application. The DQM has its web interface and some logic dealing with a database (Figure 1) that stores configuration and run quality data.

All the input data represented as multiple *parameters* that could be associated with a histogram, with a single database value, with a validity range, with an analyze *script*. Reference (good) histogram or a value can also be associated with a parameter.

These parameters also could be commented and associated with quality marks (“bad”, “user has to decide”, “doubt”, “good”, “no data” – ordered by badness). The default quality marks are retrieved in the DQM database. Then software logic and users set the actual values, a user overrides software solution.

A run quality mark is deduced as the worst of subsystem quality marks. A subsystem quality mark is the worst one for its parameters unless a mark for a special parameter “user opinion” is set. However scripts and users can mark any parameter, “user opinion” could be edited by user to override all the subsystem parameters.

The parameters are combined in presets called *configurations*. The configuration consists of clauses each one references one parameter specifying its order of displaying and whether a user can comment or assign some quality mark to a parameter. One can use the same parameter in multiple configurations to inherit users’ notes or clone it in the database to separate that notes.

The configurations form groups identified by their usage. Each *usage* serves different purpose e.g. primary check by operators, thorough check by one/another subsystem expert, the final check before data processing, etc. One *usage* combines several configurations identified by names and applied to arbitrary run range. A new configuration could be an improvement for some run range requiring special treating or just an overall update. The DQM system selects automatically the last configuration which could be applied to the run for the usage specified. However, one could use a specific configuration identified it by name.

A DQM *script* is a ROOT [7] macro implemented in C++ that produces quality information for parameter subset of one detector subsystem for one run. Each parameter requires special treating, some of them need custom quality calculation algorithms. We decided to let detector experts run arbitrary code using a simple framework instead of predefined formulae. A script has a list of arguments that could be DQM values like a histogram file path or a number. It should return a set of parameter *attributes*.

The list of parameter attributes used in DQM:

- quality mark (“bad”, “doubt”, “good”, “no data” or “user has to decide”),
- human readable comment,
- a measured/reference value to show (histogram path in a ROOT file, a computed number, a string, a null value),
- human readable parameter title.

It is not required to fill all attributes and/or all the parameters. The defaults should be used, however there could be some cases of variability like choosing the worst histogram to show, hiding a histogram or using variable parameter names for testing.

```
void script_example(
    int run, // this run number
    const char * hists // histograms ROOT file path or NULL
) {
    if(hists == NULL) {
        parameter("param1").quality(QBAD).comment("No histograms!");
        parameter("param2").quality(QBLANK).valueNull().refNull(); // unset values
    } else {
        // check the histograms somehow (e.g. by rolling dice)
        bool p1IsGood = gRandom->Integer(2), p2IsGood = gRandom->Integer(2);

        parameter("param1")
            .quality(p1IsGood ? QGOOD : QBAD)
            .comment("Checked using lazy Monte-Carlo method.");
        parameter("param2")
            .quality(p2IsGood ? QGOOD : QBAD)
            .valueHist("CL/h29")
            .refHist(32200, "CL/h29");
    }
    flush_parameters(QGOOD); // apply the changes, set script execution status
}
```

**Figure 2.** A typical DQM script structure.

Above is a ROOT macro `script_example.c` (Figure 2). Its arguments specified in the configuration. It checks if the histogram file exists and either analyzes the histograms or marks the parameters as empty or erroneous.

The script uses framework functions `parameter` for setting parameter attributes and `flush_parameters` – for exporting the attributes to the DQM and setting the script execution status. Any script is assigned to one subsystem to prevent from interference resulting from copy-paste-driven development. Since then, `parameter` accepts just a parameter name and returns a parameter object used for setting the attributes.

### 3. User Interface

We created separate interfaces for operators and experts.

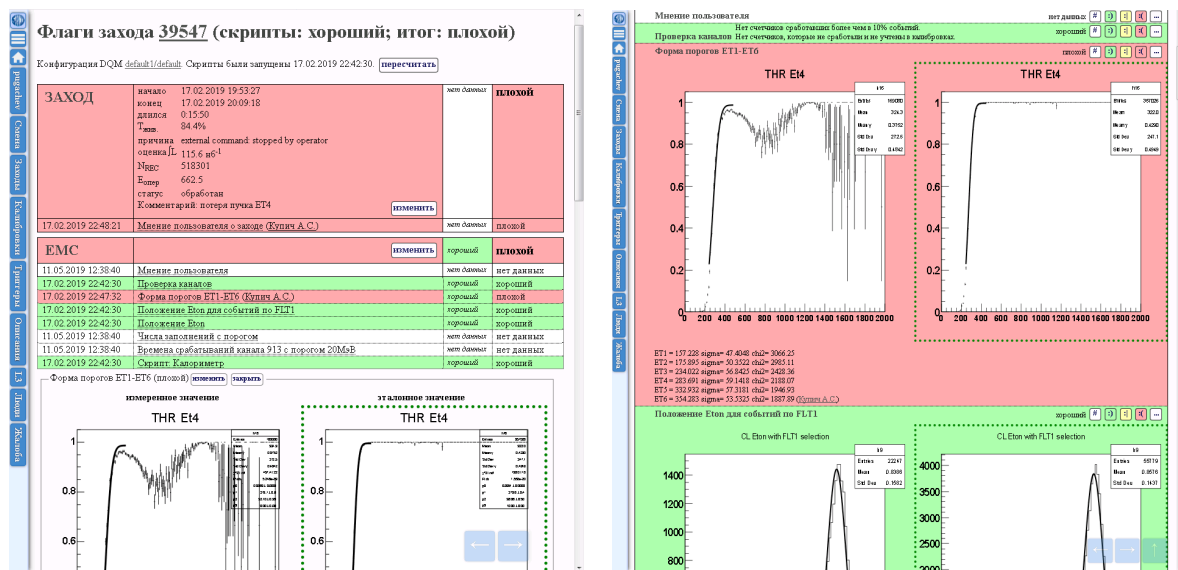


Figure 3. Reviewing a run quality – the expert mode and the operator mode.

The expert interface allows one to choose a run range, a date range or a specific shift. Then the chosen runs summary is shown. It includes some run metadata, its overall quality mark with/without user's opinion and malfunctioning subsystem list.

The hyperlinks in the “quality” column point to the run quality pages (Figure 3, the left side) including a lot of the run parameters and metadata. The parameters are displayed folded by default. An expert could expand the necessary set of ones to get more information. The histograms are shown using JSROOT [8, 9] interactive view.

Scripts are executed when a user opens web pages containing some data that depend on scripts output. A set of DQM scripts tends to run for several seconds so its results are cached. The expert mode has a button for refreshing corresponding cache item and restarting the scripts.

The operator interface (Figure 3, the right side) is aimed to simplify to perform predefined tasks for detector operators. It displays just the subsystem and parameter titles, comments and quality marks. All the parameter data are shown. An operator shall review them, leave quality marks and/or comments for special cases. The histograms are displayed as SVG figures, each parameter is accompanied with opinion buttons for quick setting quality marks.

Each parameter for each run have to be checked. Unchecked ones are marked in the DQM interface and the SND run log that is a separate tool displaying run parameters like starting time, energy, luminosity and quality marks. It takes different amount of time to process a run before DQM can analyze it. So it is very helpful to mark runs.

There is a month view showing month status for the experiment shift leader. It displays all the involved runs quality marks and theirs state of reviewing.

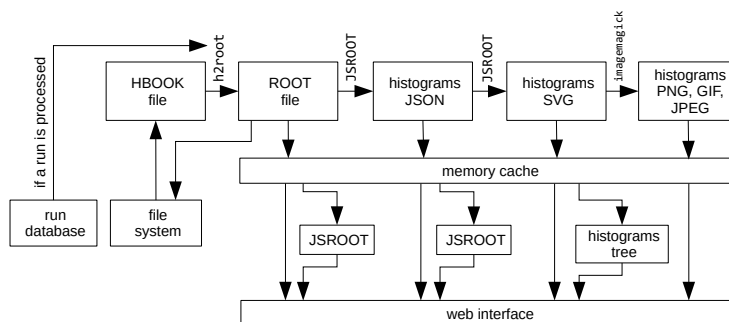
#### 4. Histogram server

In order to deliver histograms to the DQM system, a histogram server was created. It is also implemented using Node.JS. Besides having a powerful client side ROOT file browser, JSROOT can read ROOT files (`jsroot.OpenFile`), convert the objects to JSON (`jsroot.toJSON`) or SVG (`jsroot.MakeSVG`) at the server side and then build an object from JSON (`JSROOT.parse`) and draw it as an interactive figure (`JSROOT.draw`).

The server (Figure 4) is a “junction point” between the SND DQM, JSROOT, ImageMagick, h2root for older runs that use HBOOK. A user provides a histogram set name, a run number

and maybe a histogram path in the file. The server finds the proper ROOT file and responses with its path, its content or the chosen histogram. A multirun histogram viewer is implemented for experts who want to track the state changes between selected run ranges.

The reference histograms are defined for wide run ranges; examining one run means requesting several histograms from one file. So several caches holding last opened files, retrieved JSON and generated images were implemented. It results in 3-4s time delays to open a file and retrieve the first histogram and several milliseconds to retrieve others via separate HTTP requests.



**Figure 4.** The histogram server logic.

## 5. Conclusion

A new DQM system is created. It uses histograms, numbers associated with the detector parameters grouped by subsystems analyzed by ROOT macros and reviewed by users. The tool supports multiple configurations for different usages. Several views are available via web-interface. A user is able to review quality information for a parameter, a subsystem, a run, shown by default per run, shift or month.

## Acknowledgments

This work is partly supported by the RFBR grant 18-02-00382.

## References

- [1] Achasov M N *et al.* 2000 *Nucl. Instrum. Meth.* **A449** 125–139 (Preprint [hep-ex/9909015](#))
- [2] Abramov G N *et al.* 2001 *eConf* **C010430** T10 [,122(2001)] (Preprint [hep-ex/0105093](#))
- [3] Aulchenko V M *et al.* 2009 *Nucl. Instrum. Meth.* **A598** 102–104
- [4] Khazin B I (CMD-3, SND) 2010 *Nucl. Instrum. Meth.* **A623** 353–355
- [5] Bogdanchikov A G *et al.* 2014 SND data acquisition system upgrade *Proceedings, International Conference on Instrumentation for Colliding Beam Physics (INSTR14): Novosibirsk, Russia, February 24–March 1, 2014* [JINST9,C06013(2014)] (Preprint [1404.0490](#))
- [6] Nodejs Foundation 2018 Node.js" [software], version 6.14.0 <https://nodejs.org/dist/v6.14.0/> [Online; accessed 2019-05-23]
- [7] Brun R and Rademakers F 1997 *Nucl. Instrum. Meth.* **A389** 81–86
- [8] Bellenot B and Linev S 2015 *J. Phys. Conf. Ser.* **664** 062033
- [9] Bellenot B and Linev S JSROOT [software], version 5.3.1 <https://github.com/root-project/jsroot/releases/tag/5.3.1> [Online; accessed 2019-05-23]