EXTENDING PHOEBUS DATA BROWSER TO ALTERNATIVE DATA SOURCES[*]

Mihnea Romanovschi[†], Ivan Finch[1] , Gareth Howells[1]
[1]ISIS Neutron and Muon Source, Harwell Campus, Oxfordshire, OX11 0QX, UK

Abstract

The Phoebus user interface to EPICS is an integral part of the upgraded control system for the ISIS Neutron and Muon Source accelerators and targets. Phoebus can use the EPICS Archiver Appliance, which has been deployed as part of the transition to EPICS, to display the history of PVs. However, ISIS data historically has and continues to be stored in the InfluxDB time series database. To enable access to this data, a Python application to interface between Phoebus and other databases has been developed. Our implementation utilises Quart, an asynchronous web framework, to allow multiple simultaneous data requests. Google Protocol Buffer (Protobuf), natively supported by Phoebus, is used for communication between Phoebus and the database. By employing subclassing, our system can in principle adapt to different databases, allowing flexibility and extensibility. Our open-source approach enhances Phoebus's capabilities, enabling the community to integrate it within a wider range of applications.

INTRODUCTION

The Experimental Physics and Industrial Control System (EPICS) is a set of software tools and applications which provide a software infrastructure for use in building distributed control systems to operate devices such as particle accelerators, large experiments and major telescopes [1].

As part of the transition from the commercial VSystem [2] to the open-source EPICS, ISIS Accelerator Controls have deployed the EPICS Archiver Appliance [3] to record data across the control system. It has been used to archive the EPICS Process Variables (PVs) from Target Station 1 (TS1), following its recent upgrade [4].

While the EPICS Archiver Appliance offers several advantages, such as its seamless integration within the broader EPICS ecosystem and its lightweight deployment, it has limitations when applied to dynamic systems where the PV definitions are evolving. As ISIS has opted for a hybrid approach to it's transition to EPICS [5], the PV definitions are likely to change over time. For example, we recently swapped from the `NTScalar BOOLEAN` type to the `NTEnum` type for binary PVs. The current state of the EPICS Archiver Appliance does not easily provide the option to alter the data type of these PVs. The system lacks flexibility for retroactive data alterations. Its user API is challenging to integrate into novel systems, such as Machine Learning (ML) based systems. These ML systems might require asynchronous data for training, as mentioned in [6], or heavy use of statistics. These statistics would need to be placed on the ML application. In contrast, InfluxDB offers a Python-like language called Flux, accessible through its API calls for more flexible data processing and benefits from a larger open-source community compared to the EPICS Archiver Appliance [7]. Our group's preference for InfluxDB stems from the fact that we've been archiving data with it since 2019, accumulating several more years' worth of data compared to the EPICS Archiver Appliance, which we only began using for archiving in 2022. One notable advantage of InfluxDB is its capability to easily back-fill data, whereas the EPICS Archiver Appliance lacks this feature.

To address the mentioned limitations while remaining within the EPICS ecosystem, our team has experimented with substituting the EPICS Archiver Appliance with InfluxDB as the data source for the Phoebus Data Browser. This involves introducing an additional application to serve as a mediator between Phoebus Data Browser [8] and InfluxDB.

DATABASE WRAPPER

The Database Wrapper is an alternative endpoint for EPICS application requests, such as the Phoebus Data Browser, translating them to the chosen database's API, like InfluxDB, and sending the databases response in Protobuf [9] binary via HTTP to the requesting application.

The entire system is divided into four main components, adhering to a Model-View-Controller design approach (as depicted in Fig. 1):

- Phoebus Application: the view side of the system; it displays the control screens and the information from the archiving database. It also performs requests to the controller regarding what information is to be displayed.
- Quart Server [10]: an asynchronous server that acts as the controller. It unpacks the HTTP requests and delivers data from the model in an asynchronous manner. This approach minimizes the latency between processing information from a database and presenting it to Phoebus, ensuring optimal performance.
- Model: Performs the data conversion from the format that the databases API responds into the format that Phoebus expects, in this case a Protobuf binary.
- Database: for data collection, statistical analyses and metadata lookup. For example InfluxDB and CouchDB [11] respectively.

[*] Work supported by Science and Technology Facilities Council (STFC)
[†] mihnea.romanovschi@stfc.ac.uk

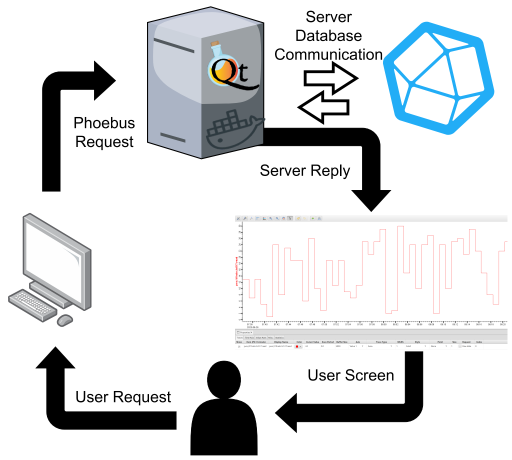


Figure 1: Overview of Architecture for the Database Wrapper applied to the Phoebus Data Browser.

Phoebus

In the context of this application, Phoebus serves as a PV viewer, enabling us to inspect and perform statistical analysis on archived PVs. When Phoebus sends a request to the archiver, it expects an HTTP response in the format of a Protobuf binary.

To showcase the functionality of the wrapper we created the PVs: ``t1halo::tc517:read'' and ``r55iwater::ambient:temperature'', which contain random values between 0-100. They exist only as entries in an InfluxDB bucket.

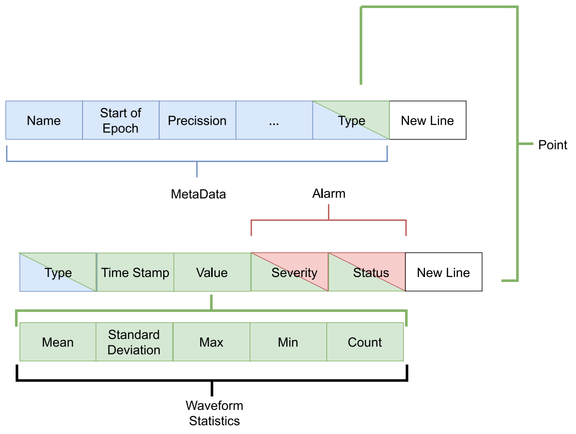


Figure 2: Overview of the Protobuf reply.

Figure 2 breaks down the format of the Protobuf binary data that the Quart server sends to Phoebus. The components are as follows:

- Header (Blue): Represents the metadata, information about the PV/Channel that we request, for example type, display settings, limits for alarms. The type is sent in both the header and body of the response. The header is separated from the body by a new line.
- Body (Green and Red): The body contains every point sampled from the archiver separated by new lines. It also contains the status and severity of any alarms that have been registered at a given time.
  - The value that is packed in the binary can be either a *scalar* type or a *waveform* type. When requesting `raw` data the request defaults to the *scalar* type when representing a PV. If `optimized` is requested then a *waveform/vector* is returned with the: mean, standard deviation, max, min, and the count of the samples used to perform the statistics. The mean will be used to plot the graph in the Data Browser when the `optimized` box is checked in Phoebus.
  - The alarm values for severity and status can come from the archiving source, InfluxDB for example, or from a different database dedicated to alarms, for example ElasticSearch.

Figure 3 is an example of a PV with random values that is archived in a test instance of InfluxDB.

Figures 4–6 showcase the difference between a `raw` and `optimized` call. By default the Database Wrapper has an aggregated of window of 30 minutes. When Phoebus makes a request it also specifies the number of points it wants to be used when computing the statistics, since the requested samples tend to be in the order of thousands the graph loses a lot of information, as can be seen in Fig. 4.

Figure 6 is the *Inspect Waveform* tool from Phoebus applied to the optimized call.

Figure 7 depicts a collection of alarms associated with the test PV ``r55iwater::ambient:temperature''. This illustrates the wrapper's capability to transmit this information by changing the relevant fields for every data point.

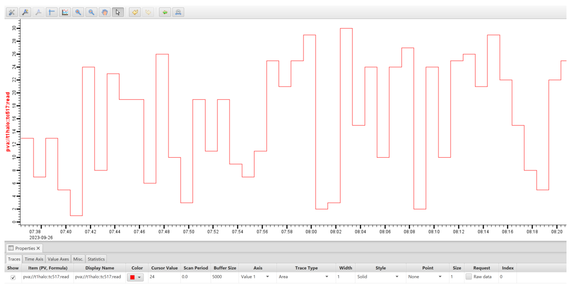


Figure 3: Example of a PV archived in InfluxDB and displayed in the Phoebus data browser.

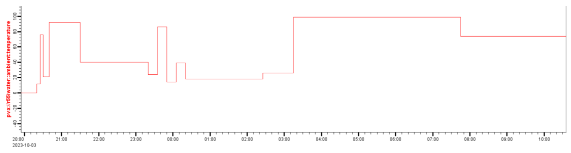


Figure 4: Example of optimized call to the Wrapper with an InfluxDB PV.

Server

The server is an application separate from Phoebus and the databases that it communicates with. It runs in a Docker

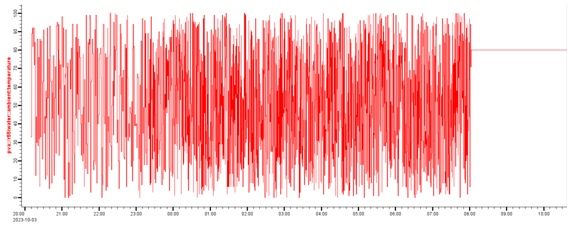


Figure 5: Example of raw call to the Wrapper with the same InfluxDB PV.

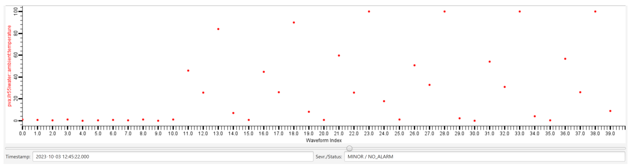
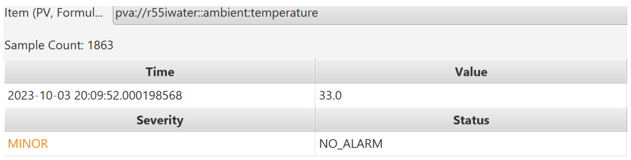


Figure 6: Example of the waveform of the optimized InfluxDB PV call.



Item (PV, Formul... pva://r55iwater::ambient:temperature

Sample Count: 1863

Time	Value
2023-10-03 20:09:52.000198568	33.0
Severity	**Status**
MINOR	NO_ALARM

Figure 7: Example of a PV with "fake" alarms. The status is **NO ALARM** but the severity is set to **0**. These parameters are set by the wrapper.

environment using the Green Unicorn HTTP Server, a platform capable of handling multithreaded operations, although this multithreaded functionality has yet to be used by the current application. Future development work will exploit the use of multi-threading to allow communications with multiple clients and data sources. Since ISIS has opted for a hybrid transition [5], having access to multiple data sources would allow us to run the EPICS Archiver Appliance at the same time with InfluxDB on the same Database Wrapper.

Our current workflow for adding a new Process Variable (PV) to be archived by a database that the wrapper can interrogate and translate is outlined as follows:

1. Define metadata parameters for the PV in the meta data database, such as name, type, precision and other elements for the display. To track this data, we use CouchDB.
2. Add it to our archiving tool, EPICS Archiver Appliance or InfluxDB

Figure 8 illustrates the internal operation of the application. It handles incoming requests from Phoebus, mimicking the EPICS Archiver Appliance API. These requests are then transformed into the appropriate API calls for the chosen database. When metadata retrieval is required, the server connects to CouchDB to retrieve the information, converting it into the Protobuf protocol. Subsequently, this metadata is incorporated as a header in the response received from InfluxDB.

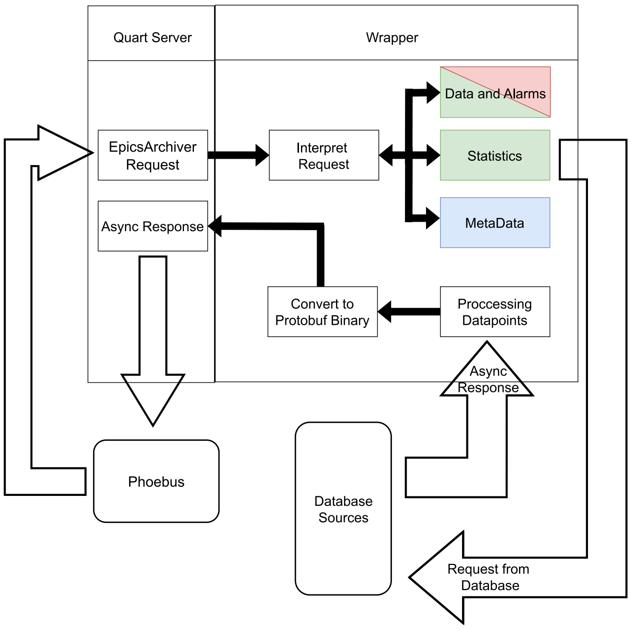


Figure 8: Overview of the server internals to form a reply.

Model

To allow for the replication of the EPICS Archiver Appliance API we have opted for an Object Oriented approach when creating the framework of the application.

The model has three components:

1. **API Interface:** defines the methods for interacting with the archive's data source, as well as methods tailored for conducting statistical analyses on the available data.
2. **MetaAPI:** allows for the selection of the payload type for the data that is collected from the archiver. It also extracts data related to settings of the PV, such as decimal precision (*PREC*) that is sent via the header.
3. **AlarmAPI:** collects the alarms from different PVs, for example ElasticSearch. When interrogated it must be matched with the reply from the API Interface. In case of missing data, for example a disconnection from the archiving database or an input/output (I/O) disconnect of the PV, timestamps might no longer match between the API Interface and the AlarmAPI, in this case a default value set in the application can be used.

Figures 9-11 offer an in-depth look at the functions that a user has to implement to be able to replicate the EPICS Archiver Appliance API. Figure 9 outlines the minimum required functionality to reproduce the results presented in this paper.

FURTHER WORK

Future development is planned to focus on three areas:

1. **Modularity:** The application currently supports only one time-series database for archiving at any given time. Using multiple databases for archiving simultaneously is not supported. We plan to make it easier to change between databases and allow for multiple data sources

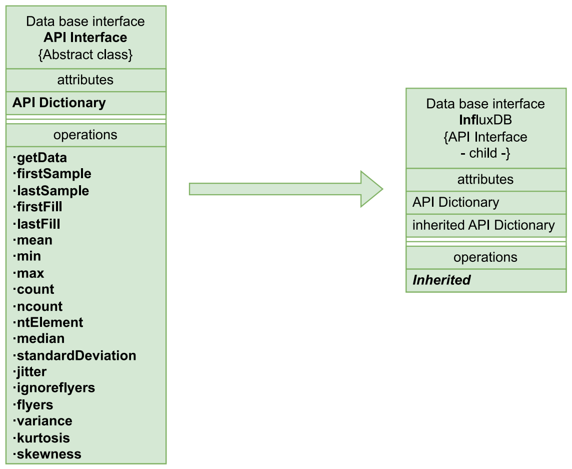


Figure 9: Overview of the methods that would be required by the InfluxDB Model in order to replicate the existing Archiver Appliance API. To replicate the results presented in Figs. 4–6, the InfluxDB Model requires the implementation of the operations: getData, mean, min, max, standard deviation, and count.

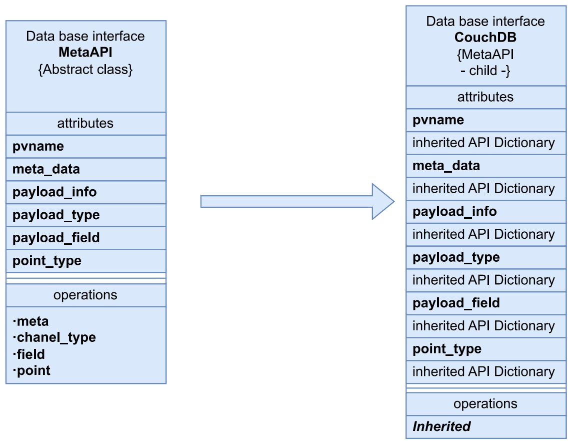


Figure 10: Overview of the methods required for the Meta-Data Model to provide instruction for the formation of the Protobuf and to correctly display in Phoebus.

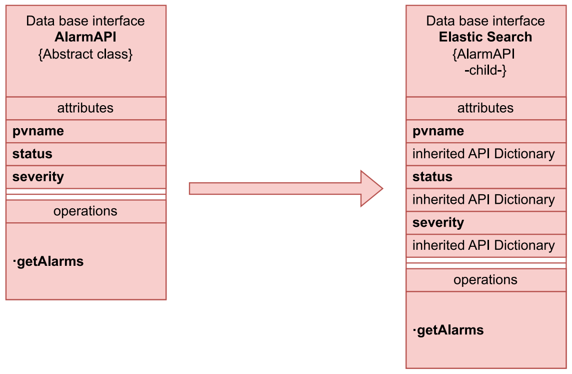


Figure 11: Overview of the alarm model. Timestamps need to be checked and aligned with the data from the archiving source, InfluxDB in this case.

to be connected for a smooth transition between systems, for example from the EPICS Archiver Appliance to InfluxDB.

2. **Alarm Handling:** As part of the transition to EPICS, ISIS accelerator alarms will be stored in ElasticSearch to support the alarm server and alarm logger applications in Phoebus. The wrapper would therefore be required to query ElasticSearch to retrieve this data.
3. **Testing:** The application requires more testing to ensure that the EPICS Archiver Appliance API is fully replicated. At present, the ability to efficiently handle simultaneous calls for PV statistics to optimize display is limited and not fully supported.

CONCLUSION

We plan to contribute to the open-source community of both Phoebus and InfluxDB through this project. This work provides an example of how the data source for the Phoebus Data Browser can be extended to databases other than the EPICS Archiver Appliance without modifying the Phoebus source code. Through it's modular design, this has expanded the options for data archiving, allowing the EPICS community to better exploit alternative, open-source tools.

REFERENCES

[1] Experimental physics and industrial control system, `https://epics-controls.org/`

[2] Vsystem, `https://www.vista-control.com/`

[3] Epics archiving appliance, `https://slacmshankar.github.io/epicsarchiver_docs/index.html`

[4] S. Gallimore and M. Fletcher, "ISIS TS1 project summary", *J. Phys. Conf. Ser.*, vol. 1021, p. 012 053, 2018. `doi:10.1088/1742-6596/1021/1/012053`

[5] I. Finch, "Progress of the EPICS transition at the ISSIS accelerators", presented at ICALPECS 2023, Cape Town, South Africa, 2023, paper TUPDP108, this conference.

[6] Tensorflow datasets, `https://www.tensorflow.org/api_docs/python/tf/data/Dataset`

[7] Influxdb time series data platform, `https://www.influxdata.com/`

[8] Phoebus, `https://control-system-studio.readthedocs.io/en/latest/intro.html`

[9] Protocol buffers: Google's data interchange format, `http://google-opensource.blogspot.com/2008/07/protocol-buffers-googles-data.html`

[10] Quart, `https://quart.palletsprojects.com/en/latest/`

[11] Apache couchdb, `https://couchdb.apache.org/`