ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-WEP41

ENeXAr: AN EPICS-BASED TOOL FOR USER-CONTROLLED DATA ARCHIVING

J. F. Esteban Müller*, European Spallation Source ERIC, Lund, Sweden

Abstract

ENeXAr is a data archival tool for EPICS-based systems. It is intended as a complement for traditional data archiving solutions, to cover use cases for which they are usually not designed: mainly for limited-duration high-data rates from a subset of signals. The service is particularly useful for activities related to machine commissioning, beam studies, and system integration testing. Data acquisition is controlled via PV Access RPC commands and the data is stored in standard HDF5-based NeXus files. The RPC commands allow users to define the acquisition parameters, the data structure, and the metadata. The usage of EPICS RPC commands means that the users are not required to install additional software. Also, acquisitions can be automatized directly from EPICS IOCs.

INTRODUCTION

Most accelerator facilities rely on data archiving services that continuously acquire and store data from all the signals that need to be monitored.

At ESS, our control system is based on EPICS [1] and we use the Archiver Appliance [2] for archiving data related to the machine operation.

There are use cases, however, when traditional data archivers are not optimal. For example, during system testing, machine commissioning activities, beam studies, or when troubleshooting issues with a particular device, users may require to acquire different signals than those regularly stored in the archiver. Often, these sets of signals contain long arrays (waveform records), which would be very costly to continuously archive for all systems, in terms of required network bandwidth and storage. However, acquiring only a subset of them for a limited period of time poses no issues.

Traditionally this is done using scripts, in which case the user needs to take care of acquiring the data using the EPICS libraries and storing the data. This results in different systems using a variety of file formats that make data analysis more complex.

The purpose of ENeXAr is to facilitate data acquisition and storage, by defining a set of commands that allow users to run an automatic data collection that saves the data in central storage, together with user-defined metadata. The only requirement for users is to have an EPICS base installation.

SOFTWARE ARCHITECTURE

ENeXAr is implemented in Python and it uses the pyepics [3] and p4p [4] packages for EPICS Channel Access and PV Access support, respectively.

Figure 1 shows a schematic view of its architecture. Clients send commands to the ENeXAr service via Remote Procedure Calls (RPC), using the PV Access protocol. There are also status PVs to monitor the service. ENeXAr processes the commands, some of which will trigger a data acquisition from EPICS IOCs. The connection to the IOCs can use either the PV Access (PVA) or Channel Access (CA) protocols.

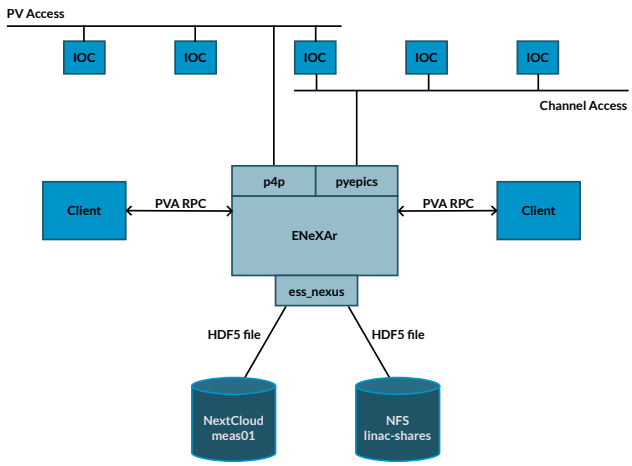

Figure 1: The architecture of the ENeXAr service. On the top, the IOCs produce data that is acquired by ENeXAr through EPICS using the pyepics and p4p epics modules. Clients send commands to the service using PV Access RPC calls. On the bottom, it is shown examples of storage backends that can be used.

The service runs on several processes using the Python multiprocessing library. The main limitation is that write operations to a single HDF5 file should always originate from the same process, since the h5py libraries do not allow for parallel write operations. As we will see later, that can be a performance bottleneck, although not very serious.

The acquisitions are then saved into the file storage backend. The data is formatted using the NeXus [5] convention and saved as HDF5 [6] files. Files are created using the ess-nexus Python package [7]. Files are not indexed and the directory structure inside the storage backend is completely managed by the users.

Multiple instances of ENeXAr can be deployed in the same network, provided that they use a different prefix for the PV names used for commands and status. That allows, for example, for each instance to use a different storage backend, or as a means for balance the load in the service.

USAGE

As stated above, the user controls the data acquisition via EPICS RPC calls, which can be generated by any EPICS7-

* JuanF.EstebanMuller@ess.eu

ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-WEP41

compatible client. One option is using the `pvcall` command from the command-line. The procedure names are formed by appending the command to the prefix that identifies the ENeXAr service. There is also a Python module available, called enexar-cli [8], which simplifies the usage of RPC calls from Python and also provides a JupyterLab graphical interface. Finally, calls can also be generated by other IOCs.

Below one can find the list of commands available, grouped by function, and with a short explanation.

Acquisition-Related Commands

To start and stop acquisitions, the following procedures are available:

- `ACQUIRE`: start the acquisition of a PV into a file. It could be a continuous measurement that is manually stopped later on, or only for a user-defined number of PV updates. The arguments to this call are the PV name, the file path, acquisition type, and optionally a set of metadata. It also supports EPICS filters for data reduction on the IOC side.
- `ACQUIRE_S`: synchronous version of the `ACQUIRE` command that will return only after the measurement is done. Only useful for short acquisitions.
- `STOP`: stop the acquisition of a given PV into a file. Other acquisitions to the same file will continue.

File-Related Commands

These are commands that determine the way ENeXAr deals with files. They don't correspond to the way files are managed by the operating system.

- `OPEN`: this command is required if the user wants to append data to an already existing (and closed) file. If the file does not exists, it is created, although in that case this command is not needed and the file can be created using the `ACQUIRE` command.
- `CLOSE`: closes a file and does not allow further acquisitions to be appended into the file, unless the `OPEN` command is used. If there is any acquisition running, they are stopped.
- `LS`: return the output of running the `ls` command on the path passed as a parameter.

Status-Related Commands

The following operations are available for users to check the status of the acquisitions and files managed by an ENeXAr instance:

- `STATUS`: prints all the acquisitions that are taking place.
- `IS_OPEN`: check if a file is open for appending data.
- `IS_ACQUIRING`: checks if a PV is being acquired into a given file.

INTEGRATION IN JUPYTER

At ESS, we use JupyterHub [9] to develop and run Python scripts that interface with the accelerator control system for commissioning and operations-related activities. This infrastructure also allows users to perform data analysis on the cloud.

In our configuration, the same storage backends that are used by ENeXAr to save data, are also mounted on JupyterHub to enable easy access to the users of the data.

The enexar-cli Python module delivers also a graphical user interface, which is built based on Jupyter Widgets [10]. The UI allows users to graphically start and stop acquisitions, as well as to check the service's status. Figure 2 shows a screenshot of the interface, which is based on another existing UI called "PV Saver" that was used to take data acquisitions directly into raw HDF5 files. This UI can be run using Voilà [11] to hide all Python code and make it more user-friendly.

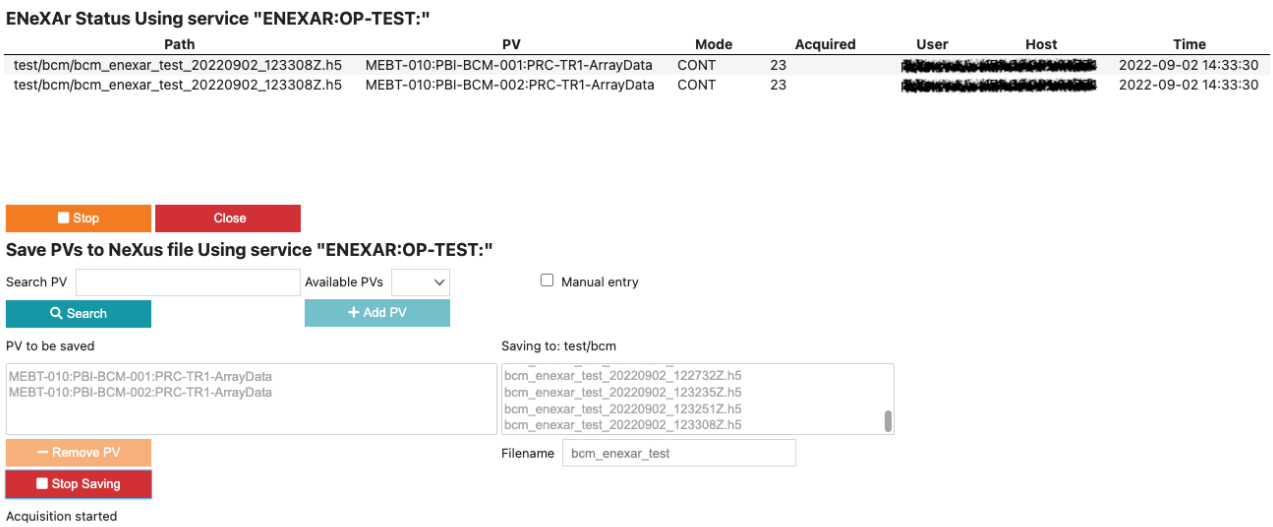

Figure 2: Screenshot of the Jupyter interface to ENeXAr, using Voilà [11] to render the UI. On the top, it shows the acquisitions that are currently taking place and on the bottom, the user can start new measurements.

ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-WEP41

WETEST INTEGRATION

One of the use cases of ENeXAr is to take acquisitions from IOCs while running system integration tests. These acquisitions can be used for calibration purposes or as part of the testing process.

At ESS, the Beam Diagnostics group relies on WeTest [12] for test automation. ENeXAr has been integrated in a forked version of WeTest to make possible this use case. An example of how to use it is shown in Fig. 3.

```
tests:
  - name: "Test 1"
    setter: "PV:set"
    commands:
    - name: "0"
      set_value: 0
      get_value: 0
    logger:
      - pv: "PV:TO:BE:ARCHIVED"
        server: "ENEXAR:PREFIX:"
        path: "measurements/test"
        acquisitions: 1
        metadata:
          file_user:
            name: "juan"
            affiliation: "ESS"
          file_entry:
            title: "Test"
            description: "Testing logger"
          instruments:
            INSTRUMENT_1:
              nxtype: "NXbeam_instrumentation"
              name: "Instrument name"
```

Figure 3: Snippet of a WeTest example test case using ENeXAr to perform an acquisiton.

BENCHMARK

The performance of ENeXAr was studied in a series of tests. Two virtual machines (VMs) were used for the tests, one to run a virtual soft-IOC, and another one running ENeXAR with the same storage backend as the one used in production. The status of the VMs and the virtualization cluster were monitored during the tests. In addition, we made sure that the two VMs were running on different hosts in the virtualization cluster, so that the data would be sent through the network.

The soft-IOC can be configured to generate a certain number of PVs, each of them with a given array length. During the tests, the number of PVs is varied, and the array length is adjusted to match the desired data rate. For our tests, the IOC was running at an update rate of 14 Hz, which is the pulse repetition rate of the ESS linac.

The first set of tests consisted of the acquisition of 1, 10, 20, and 50 PVs, storing each PV in a separate file, and sweeping the data rate from 100 Mbps to approximately 7.5 Gbps. Results are presented in Fig. 4, and they show that ENeXAr is able to cope with data rates of up to 2.5 Gbps, except for the case when only 1 long array is archived.

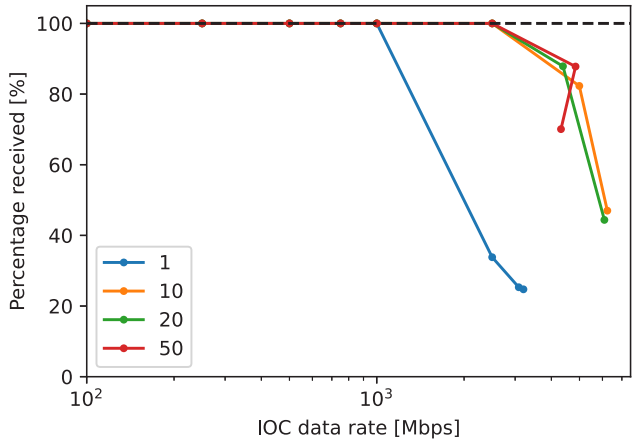

Figure 4: Storage success rate for different data rates produced by a single IOC and stored in one file per PV. The lines correspond to the number of PVs generated by the IOC, for which the array length is adjusted to match the desired data rate.

For the second test case, the same set of tests were run with the only difference that ENeXAr was configured to store all PVs in the same file. Figure 5 shows the results, which are slightly worse than in the previous case. For this configuration, the limit seems to be around 1 Gbps.

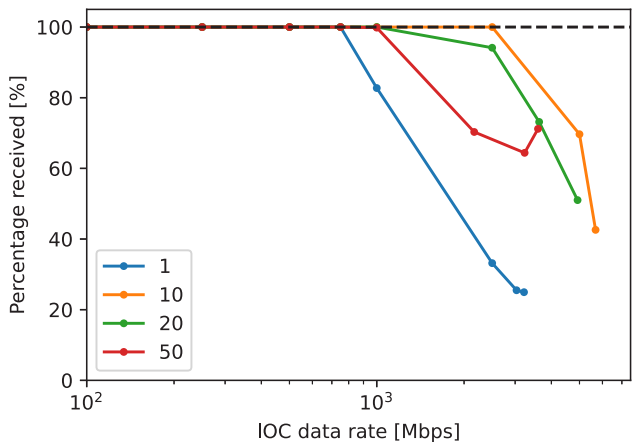

Figure 5: Storage success rate for different data rates produced by a single IOC and storing all PVs in the same file. The lines correspond to the number of PVs generated by the IOC, for which the array length is adjusted to match the desired data rate.

In the first case, each file is written to the storage backend from a different Python process. The results are compatible with the bottleneck being the storage backend. In the second case, all the data is written to a file from the same process, and the performance could be limited by the writing speed of the HDF5 libraries.

Finally, the same test cases were run using the PV Saver script, which is the tool that has been widely used at ESS for dedicated data acquisition. The results are presented in Fig. 6 and show a lower performance. Only 2 test cases achieve a 100% storage success rate, for the lowest data rate of 100 Mbps. This can be due to a combination of

ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-WEP41

two factors: PV Saver runs on a single process and it uses Jupyterhub's storage backend, which seems to be slower than the one used by ENeXAr.

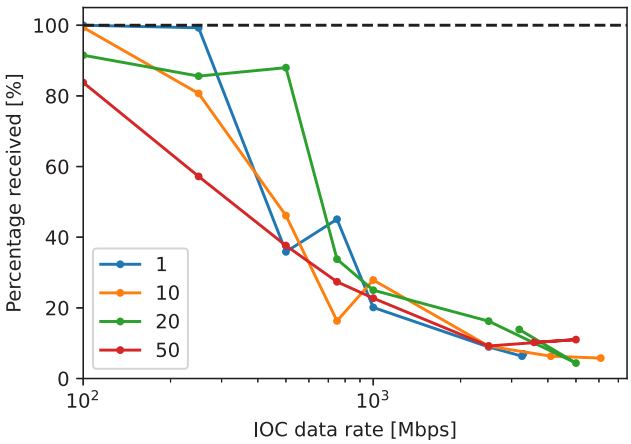

Figure 6: Storage success rate for different data rates produced by a single IOC, using the PV Saver script on JupyterHub. The lines correspond to the number of PVs generated by the IOC, for which the array length is adjusted to match the desired data rate.

SUMMARY

In this paper, the ENeXAr service was presented, which is an EPICS-based tool intended to simplify on-demand data acquisition from IOCs. Its architecture and usage were also described. Finally, results from performance tests were presented, showing a significantly better performance than the previously available alternatives.

The code publicly is available on `https://gitlab.esss.lu.se/ics-software/enexar` under the GPLv2 license.

ACKNOWLEDGEMENTS

Thanks to Emanuele Laface for the development of the "PV Saver" UI, which somehow triggered the development of ENeXAr and was used as a base for its Jupyter interface. Thanks also to Stephane Armanet and Alessi Curri for their support in setting up the test environment.

REFERENCES

[1] EPICS website, `https://epics-controls.org/`

[2] Archiver appliance website, `https://slacmshankar.github.io/epicsarchiver_docs`

[3] M. Newville, `https://pyepics.github.io/pyepics/`

[4] M. Davidsaver, `https://mdavidsaver.github.io/p4p/`

[5] NeXus data format website, `http://www.nexusformat.org/`

[6] HDF data format website, `https://www.hdfgroup.org/solutions/hdf5/`

[7] N. Milas and C. Derrez, *ESS NeXus repository*, `https://gitlab.esss.lu.se/ess-bp/ess_nexus`

[8] J. F. Esteban Müller, *ENeXAr-cli repository*, `https://gitlab.esss.lu.se/ics-software/enexar-cli`

[9] JupyterHub website, `https://jupyter.org/hubs`

[10] Jupyter Widgets website, `https://ipywidgets.readthedocs.io/`

[11] Voilà website, `https://voila.readthedocs.io/`

[12] WeTest repository, `https://github.com/epics-extensions/WeTest`