A PYTHON FRAMEWORK FOR INTEGRATION OF MEASUREMENTS INTO THE EPICS CONTROL SYSTEM

P. Schreiber*, E. Blomley, J. Gethmann, M. Schuh, A.-S. Mueller
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

Many measurements in accelerator physics require dedicated scans of parameters, such as the main frequency of the RF system for chromaticity measurements or a variation of quadrupole strength for beta function measurements, etc. Such measurements cannot be performed by simply reading an instrument, but require a certain measurement procedure. These measurements are typically implemented as scripts, as they are often written by staff or students whose task is the measurement itself. For reusability and maintainability, however, an integration into the EPICS control system is favoured. This has multiple benefits such as the easy archiving of results in the central database or the integration of the measurements into the standard operator panels. To allow non expert staff and students to easily build measurements exposed via the EPICS control system, we built a framework in Python to implement such measurement routines as EPICS input-output-controllers (IOC). This framework allows the author of such measurements to focus on the measurement itself and still benefit from an EPICS integration without the need for extensive knowledge in EPICS IOC development.

INTRODUCTION

Operation of an accelerator facility for user operation as well as experimental science, often requires measurements of certain quantities. Measurements depending on a single device reading are easy to do and reuse, however, once a measurement requires a measurement routine it is more complicated to implement maintainable and reusable routines. While standard measurements, required for every-day operation are usually well integrated in the operation procedures and control systems, more specialised measurements are often developed as scripts, making them sometimes hard to find and reuse.

Integrating measurements into the control system solves this problem. It usually requires a certain understanding of the control system. This requirement is not always fulfilled for students, guest scientists and scientists, therefore, it is difficult for them to implement the integration themselves.

Therefore, the IBPT measurement package implements a framework written in Python [1] to easily write measurements with automatic EPICS [2] integration. It does not require much pre-knowledge and uses pythonic ways to achieve this goal. In addition to the EPICS integration it offers further interfaces as described below.

* patrick.schreiber@kit.edu

MEASUREMENTS

Measurements are intended to be executed multiple times, sometimes within the same process (in the case of IOC integration, as described below). They are implemented by sub-classing a base measurement class that provides the required structures and takes over the integration into the control system. Observables and actuators are implemented as subclasses of the "Monitor" and "Actuator" classes that handle metadata collection and clean-up automatically and are registered to the measurement class. The execution of a measurement is split into three steps:

1. Preparation,
2. Measurement,
3. Post Measurement.

The first step, preparation, allows the code to prepare the measurement each time it is executed. Tasks for this could include gathering general data or setting-up some device. It is not entirely required to implement this, all the steps could be implemented in the second step, but it helps with code maintenance by clearly separating preparation from execution. The second step, the actual measurement, is where the measurement loop should be implemented. It is called right after the preparation step. The loop for executing procedures, e.g. scans, is intentionally left to the user, as having too much forced structure would lower acceptance with newly arriving people. This step can be interrupted in a controlled way when using "break points"[1]. The last step to implement is the post measurement procedure. This is where the analysis of the measured data is happening. The return value of this method must be a dictionary that will be saved to disk.

Preparation

In addition to the usual user implemented preparation, the base measurement will do it's own preparation. All registered monitors and actuators will prepare metadata collection. Actuators will also record the current value and use that for resetting after the measurement is aborted or finished.

Measurement

The measurement method should use the registered actuators to modify the system for the measurement (if required). It should use the "read_data" method to read all the values from actuators and monitors. By using this method, the data, together with metadata is gathered in a data store, that will be saved to disk in any case, even if the measurement fails or

[1] Markers indicating a breakable point, not to be confused with debugging breakpoints

is aborted. This prevents data loss and expensive repetitions of measurements if e.g. there is a bug in the user's measurement routine leading to an Exception. Measurements can yield `True` to indicate a breakable point where the measurement can be aborted or use a sleep method from the base class that acts as a breakable sleep where at any point the measurement can be stopped.

When a measurement is aborted or throws an Exception, all actuators will be reset to their initial value. If more complex reset behaviour is required, overriding the "cleanup" method of the "Actuator" class allows for custom code.

Post Measurement Procedures

To do the data analysis the third method is used. If configured, it can be run on failed or aborted measurements or just when the measurement method exited successfully. The return value of this method is put into the resulting file on disk in an HDF5 Datagroup[2] called "Results".

All the data gathered with "read_data" and returned from this method will be saved in an HDF5 [3] file. If the data is very large or cannot be lost, it is possible to configure the measurement to save data immediately after creating it. Otherwise it will be saved to disk at the end of the post measurement method. Some measurement devices directly produce files instead of in-memory data (e.g. due to the amount of data). If these are used for a measurement, the functionality can be implemented as a subclass of a provided monitor class for such a case. This class is recognized as special monitor by the framework. The resulting data file will contain the filename of the created files. The entire collection of files and normal result file will be packed into a single large zip-file. In case the device produces the same filename each time, the filenames will be suffixed with the unix timestamp.

The normal result file has three data groups, one for data and metadata from actuators, one for data and metadata from monitors and one for the results.

Great care is taken to ensure that any fails of user implemented methods will result in saved data until that point to ensure no data is lost. The base class also takes care that the system is left in the same state as it was before.

CONFIGURATION

Most of the time a measurement will have some configuration options. The measurement framework allows to specify such configuration options. Each option must specify a name, a default value, a data type and optionally a short help text. The data type is given as a Python type with the exception of Enum. These values ensure that the various interfaces, provided by the framework, can expose these options. Having these configuration options centralised also allows to save the configuration in the result file.

Furthermore, the framework allows to configure the output of measurements and results to be configured similarly to the configuration options. This allows to present the output in the different interfaces in the correct manner, e.g. creating the correct Process Variables (PV) for the EPICS interface. This also allows to specify only a subset of the resulting data as "output" for the interfaces while the rest of the data is still saved in the data file.

[2] Hierarchical elements in HDF5 files used to group datsets.

INTERFACES

The measurement framework comes with multiple possible interfaces. By using the configuration options described before, these interfaces are auto generated without the need to manually define them.

Command Line

The main entry point that is used to start the other interfaces is the command line. It serves two purposes. One is to execute a measurement directly from command line by passing the configuration options as command line arguments. The second is to allow to specify the interface to use if one of the other interfaces is desired. It offers a help flag that prints some general help for measurements and also the descriptive text for each configuration option if it was given in the configuration set up.

GUI

A GUI can be started that will show type-specific input widgets for the configured configuration options as well as fields for the generated output from the measurement itself and the results from the post measurement procedure. It also contains general control widgets such as buttons to start and abort a measurement and some status information including the log output of the measurement. Using this interface it is possible to run the measurement multiple times without restarting the process or reopening the GUI.

An example for the auto generated GUI for a demonstrator implementation of a chromaticity measurement is shown in Fig. 1.

EPICS IOC

The measurement can run as a long serving process on a server in form of a Python SoftIOC [4]. It offers PVs for configuration of the measurement as well as PVs for returning the results from either the measurement or the post measurement routine. It is possible to specify for which results a PV is generated when configuring the output. Further PVs always present include PVs for starting and aborting the measurement, the status of a measurement as well as the log output of the measurement

In the measurement method, it is possible to update the value of outputs iteratively by calling a method and specifying the given name of an output as well as the value. This will directly update the value of the associated PV, which can be used, e.g., to show a live progress.

Panels The framework offers to automatically build CSS and Phoebus panels [5]. It infers the type of widget to use from the configured data types for configuration options and outputs. Furthermore, for Phoebus, it is possible

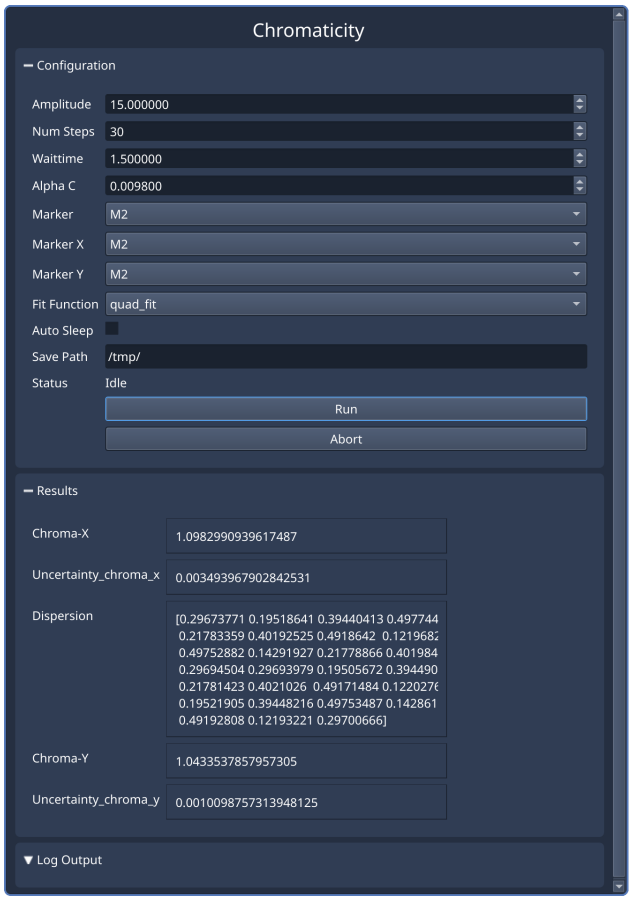

Figure 1: Example auto generated GUI for a demonstrator implementation of a chromaticity measurement.

to configure plots with a dictionary containing keys for labels, names for x and y data sources (as names of configured outputs) as well as how to combine multiple lines into one and some limited styling. These plots will be placed automatically in the panel. An example panel for a chromaticity measurement is shown in Fig. 2.

OUTLOOK

A number of additional features are foreseen to be implemented in the near future. Among these is to strengthen the integration into the research data management at IBPT [6] where the resulting files are automatically pushed to the Kadi [7] repository including metadata for easy discovery of measurement results. Also, the automatic aggregation of multiple measurements (potentially automatically run back-to-back) into average and standard deviation values for better results is planned. Furthermore, an option to write simple scanning measurements with minimal or without any Python code is under consideration by only using configuration files such as YAML files [8].

SUMMARY

A framework to implement Python measurements with explicit automatic integration into the EPICS control system has been shown. It allows users without knowledge of the control system to make their measurement implementations reusable and maintainable. The framework takes care of saving the data, generating panels, and gracefully handling aborts and exceptions. It can be used from command line, as an EPICS IOC or with a GUI, where all these interfaces will be automatically generated. Therefore, interfaces from different measurements follow the same general philosophy making them recognizable by operators.

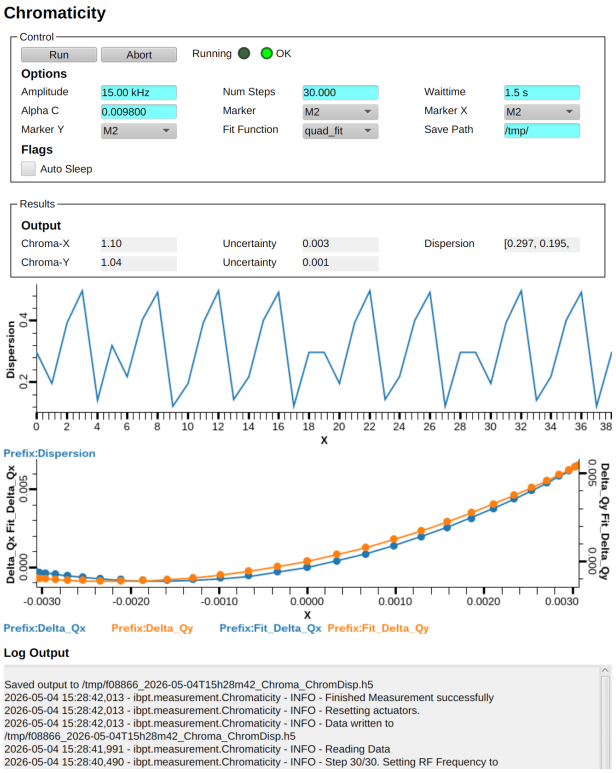

Figure 2: Auto generated Phoebus panel for a chromaticity measurement implemented using the measurement framework. The top part labelled "Control" is defined by the general control widgets and the auto generated widgets based on the configuration parameters of the measurement. The second part labelled "Results" is defined by the auto generated widgets based on configured output parameters. The plots show the measured dispersion (upper plot) and measured tune changes as well as the fit for the chromaticity determination (lower plot).

ACKNOWLEDGEMENTS

This project has received funding from the European Union's Horizon Europe Research and Innovation Programme under Grant Agreement no. 101057511 (EURO-LABS).

REFERENCES

[1] Python Software Foundation, "Python", https://www.python.org

[2] EPICS Control System, https://epics-controls.org

[3] The HDF Group, "Hierarchical Data Format", https://www.hdfgroup.org/HDF5/

[4] Python SoftIOC, https://github.com/DiamondLightSource/pythonSoftIOC

[5] Control System Studio, https://www.controlsystemstudio.org/

[6] S. Masoumi, "Self-Hosting Research Data Infrastructure with Kadi4Mat: A Practical Use Case for Managing Physics Data at IBPT, KIT", in *Data Stewardship goes Germany (DSgG)*, Sep. 2025. doi:10.5281/ZENODO.17224843

[7] N. Brandt *et al.*, "Kadi4Mat: A Research Data Infrastructure for Materials Science", *Data Sci. J.*, vol. 20, p. 8, Feb. 2021. doi:10.5334/dsj-2021-008

[8] YAML, https://yaml.org/