ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM010

SECoP INTEGRATION FOR THE Ophyd HARDWARE ABSTRACTION LAYER

P. Wegmann*, K. Kiefer, W. Smith, L. Rossa, O. Mannix, HZB, Berlin, Germany
M. Zolliker, PSI, Villigen, Switzerland
E. Faulhaber, TUM FRMII, Munich, Germany

Abstract

At the core of the Bluesky experimental control ecosystem the Ophyd hardware abstraction, a consistent high-level interface layer, is extremely powerful for complex device integration. It introduces the device data model to EPICS and eases integration of alien control protocols. This paper focuses on the integration of the Sample Environment Communication Protocol (SECoP) into the Ophyd layer, enabling seamless incorporation of sample environment hardware into beamline experiments at photon and neutron sources. The SECoP integration was designed to have a simple interface and provide plug-and-play functionality while preserving all metadata and structural information about the controlled hardware. Leveraging the self-describing characteristics of SECoP, automatic generation and configuration of Ophyd devices is facilitated upon connecting to a Sample Environment Control (SEC) node. This work builds upon a modified SECoP-client provided by the Frappy framework, intended for programming SEC nodes with a SECoP interface. This paper presents an overview of the architecture and implementation of the SECoP-Ophyd integration and includes examples for better understanding.

INTRODUCTION

Moving and integrating research equipment between facilities with different Experimental Control Systems (ECS) can be a challenging and time-consuming process. Sample environment hardware, in particular, is usually not permanently attached to a specific experiment and is often moved both within and between research facilities. The Sample Environment Communication Protocol (SECoP) [1] has been developed under direction of the International Society for Sample Environment (ISSE) [2] to facilitate this process. It is also intended as an overarching solution for standardising communication with sample environment equipment at photon and neutron research facilities. The core design principles of the protocol (i.e. simple, inclusive, self-describing and providing rich metadata), were chosen to achieve this goal. In particular, the self-describing properties and the inclusive design philosophy behind SECoP facilitate its integration into various experimental control systems. This is also leveraged in this publication for the integration of SECoP into the hardware abstraction layer Ophyd [3], with the effect that Ophyd devices are constructed automatically upon connection, whilst retaining all metadata about the controlled hardware. A crucial application of this SECoP-Ophyd integration we introduce here is to facilitate heterogeneous configurations, integrating fixed beamline equipment supported by EPICS with mobile sample environment equipment supported by SECoP in the same Bluesky [4] environment.

SECoP Hardware Abstraction

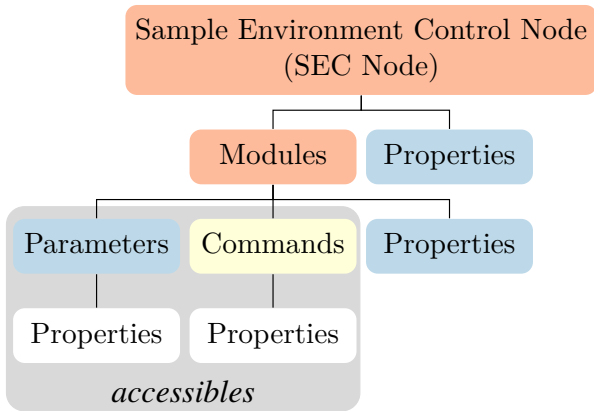

Figure 1: Schematic of SECoP hardware abstraction.

According to the SECoP specification [5], the access point for a SECoP device is a SEC node. The SEC node thereby acts as a server that allows connections from an arbitrary number of clients and gives access to the published functionalities of sample environment equipment. Within the SEC node, physical quantities of a sample environment device are logically represented by *modules*. The term *module* has been deliberately chosen to distinguish it from a device, as a sample environment apparatus is often composed of a number of modules and is only called a device as a whole. All modules are composed of accessibles, which can either be *Parameters* or *Commands*. While parameters provide live information on modules, commands are provided to initiate specific actions of a module. This internal structure is depicted in Fig. 1. At every level, static information of the parent entity is provided by property fields. They ensure a structured way of holding metadata regarding the SEC node and hardware it is connected to. For example, at the parameter level, the mandatory *datainfo* property holds important information about the datatype, unit and other metadata. Furthermore, the SECoP standard defines interface classes for modules, with predefined functionality and the meaning of specific mandatory accessibles and properties. Most notable here are the *Readable* and *Drivable* interface classes, enabling a standardised way of constructing readable modules such as a simple temperature sensor and drivables such as

* peter.wegmann@helmholtz-berlin.de

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM010

a controlled temperature or a magnetic field. This allows an ECS to determine the functionality of a module directly from its description and configure itself accordingly.

Ophyd Hardware Abstraction

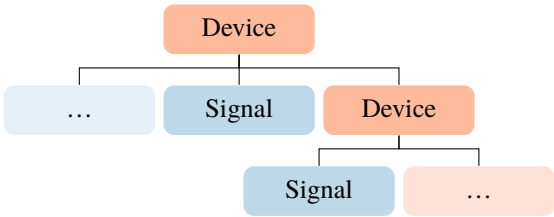

Figure 2: Tree-like structure of an Ophyd *Device*.

In Ophyd hardware is represented as a *Device*, which is hierarchically composed of sub-devices and *Signals*. This means that an Ophyd *Device* has a tree-like structure of *Devices* as nodes and *Signals* as leaves. This tree can have an arbitrary depth as shown in Fig. 2. A *Signal* represents an atomic variable within the Bluesky ecosystem that cannot be further decomposed by layers above Ophyd. *Signals* can be either read-only, write-only or read/write, depending on the capabilities of the underlying hardware being represented. In addition, each *Signal* has a *describe()* method that returns all available metadata for the *Signal*, including mandatory fields such as *dtype* (JSON type of the *Signal*), *shape* (specifying whether the transported data is a scalar or an array, including exact dimensions of the array), and *source* (identifier for the signal, e.g. EPICS process variable). Some optional fields like *units* or *precision* are also specified , but custom fields can be added at will. Within a device, the *Signals* are grouped into read signals and configuration signals.

The supported interfaces of a *Device* are realised in a modular way by utilising *mixin* classes. This allows for a flexible way of constructing Ophyd devices with varying complexity and functionality. The minimum interface required is the *Readable* interface, which permits a formalised way of reading and retrieving metadata of all read signals. Other mixins, such as *Movable*, enable interfacing with slow-moving devices like a temperature controller.

Integration of Other Control Systems Into Ophyd

Since Ophyd does not come with any device drivers to directly interface with hardware, another layer is usually needed underneath it. Currently Ophyd is mostly used to integrate with EPICS-backed hardware, but various packages for interfacing with other control systems have been published, of which the following are the most relevant:

- `yaqc-bluesky` [6]: A bluesky interface to the yaq [7] instrument control framework.
- `tango-ophyd` [8]: Experimental and incomplete integration for the Tango control system.
- `pycertifspec` [9]: Integration for the SPEC instrument control and data acquisition software.

The yaq project stands out here since it is very similar to SECoP in some regards. The standard was built to be self-describing, simple, portable and reusable, but lacks focus on rich metadata. Much like the packages presented above, the integration SECoP-Ophyd facilitates the creation of Ophyd devices representing connected hardware components. But in contrast to the other integrations, generated *Devices* are not atomic. Meaning that they are more complex and composed of sub-devices with interdependencies between them.

SECoP-Ophyd also differs in another aspect from the presented packages in that it builds on the not yet officially released `Ophyd.v2`. API that utilises the Python asyncio library, which is also at the core of the Bluesky Run Engine. Even though Ophyd.v2 is not officially released yet, the interface is already reasonably stable, and use of the new API was chosen because integrating control systems is made easier by only demanding a narrow backend interface. Another key advantage is that support for SECoP commands can be ensured by the SignalX Class introduced in Ophyd.v2.

The implementation of the SECoP-Ophyd integration described here heavily relies on a SECoP-client provided by the Frappy-package [10]. The Frappy-client offers an already mature interface and is responsible for handling the connection and all SECoP communication to the SEC node.

ARCHITECTURE OVERVIEW

An overview of the software architecture used for the SECoP-Ophyd integration is given in Fig. 3. On the right side, a SEC node as an interface to a logical device is shown, which may be composed of several physical hardware components. Together they can be regarded as mobile sample environment equipment (marked in green). Within the SEC node, the components of the equipment are represented as individual modules. In the centre of Fig. 3 the SECoP-Ophyd integration is shown in yellow. At its core is the *SEC node Ophyd.v2* Device and the Frappy-client. Structurally the SECoP-Ophyd integration mirrors the device structure predefined by the SEC node. A clear separation of concerns is achieved by having the SECoP-client act as backend for the Ophyd.v2 devices handling the connection and SECoP communication to the SEC node. In order to make the Frappy SECoP-client compatible with Ophyd.v2, it was wrapped in a new class, ensuring that requests to the SEC node are non-blocking.

The *SEC node Ophyd.v2* Device acts as an entry point and initialises the SECoP-client object upon construction. The constructor for the *SEC node Ophyd.v2* Device has a very simple interface and requires only the IP address and port number of the SEC node. Once the client has established a connection to the SEC node and parsed the answer to the *describe* message (for more information on the *describe* message see [1]) it has sent to the SEC node, child devices are created for each module present within the SEC node. These devices all provide the interface of a standard readable or movable device, which is expected by the Bluesky run

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM010

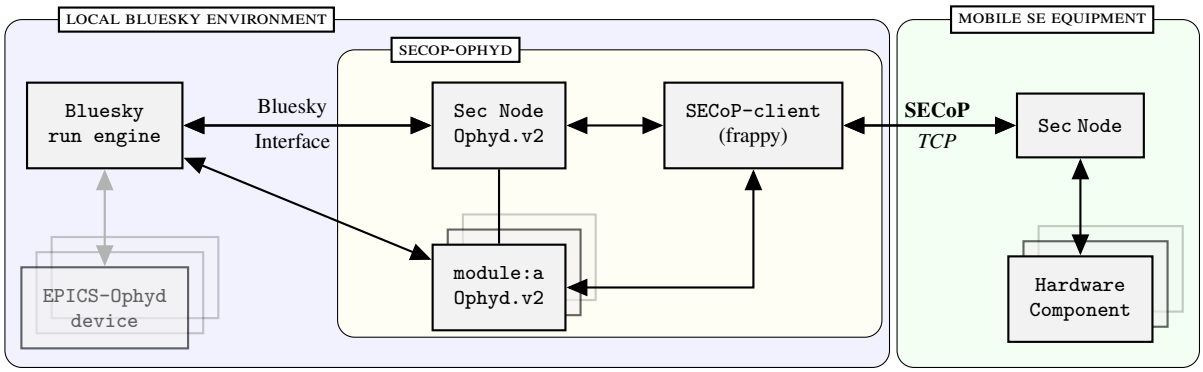

Figure 3: Architecture overview of how the SECoP-Ophyd integration interfaces with the Bluesky run engine and a SEC node.

engine. This means that they can be easily integrated and operated alongside other Ophyd devices that are connected to hardware via EPICS or any other underlying control system. This is indicated by the greyed out EPICS-Ophyd device under the Bluesky run engine.

HARDWARE ABSTRACTION CONVERSION FROM SECOP TO OPHYD

The goal was to have direct modelling of a SEC node and all attached modules, properties and accessibles as an Ophyd *Device*. This is facilitated by the concise and machine-readable information published by all given SEC nodes. The colourisation in Figs. 1 and 2 indicates the chosen correspondence between the different concepts of representing hardware. As implied by the red colour, SEC node and attached modules are modelled by Ophyd *Devices*. Respectively hinted in blue, all *Properties* pertaining to the SEC node and modules, as well as *Parameters*, are turned into *Signals* in the final Ophyd *Device*. All metadata that was held in the *Properties* of a given *Parameter* is still accessible via the corresponding *Signals* describe method.

SECoP Drivable Modules

One crucial feature of SECoP and Ophyd is their abstraction of slow settable values. The SECoP *Drivable* interface class requires modules to have a *target* and a *status* parameter, in addition to an argument-less *stop* command. As soon as a new target value is received by the SEC node, the status value switches from an *IDLE* state to a *BUSY* state, and it goes back to *IDLE* when the target is reached. There are further possible state transitions the module can take, but this is the simplest sequence of state transitions. *Drivable* modules in SECoP correspond directly to *Movable* devices in Ophyd. The interface for a *Movable* only requires a *set()* method, with the new value as an argument and a Status object as a return value, which is marked done once the device has finished moving. In the case of Ophyd.v2, the returned Status is an *AsyncStatus* that can be awaited. Additionally, when the optional *Stoppable* mixin is added, a functionally complete representation of a *Drivable* is achieved within Ophyd. If the *set()* method of a *Movable* Ophyd device is invoked, the SECoP-Ophyd integration internally transmits a SECoP message to the SEC node to update the desired target value and returns an *AsyncStatus*. Upon construction of the *AsyncStatus*, an asyncio Task is started, that monitors the status parameter of the driven module. Once the status changes from *BUSY* to *IDLE*, the Task finishes and the AsyncStatus is marked done.

SECoP Commands

SECoP commands are treated in a special way, as there is no equivalent concept in Ophyd. At first glance, the SignalX class recently introduced in Ophyd.v2 looks promising. It implements the command design pattern [11], which means that these signals provide an `execute()` method without any input parameters for triggering specific actions. However, SignalX alone is inadequate for modelling commands that have multiple arguments and return values. Because of this limitation, it was decided to encapsulate commands in sub-devices. Each sub-device contains signals that can be read and written for each argument, as well as signals that can only be read for each return value. There is also a SignalX signal to initiate the execution of the command. Furthermore, the signals related to the return values are configured as read signals of the device, while the other signals are set as configuration signals. To execute a SECoP command, the argument signals must first be assigned values. Then the `execute()` method of the SignalX signal is called. Internally, the SECoP-clientprogram then sends the corresponding command to the SEC node. As soon as the response is received by the SECoP-client, the values are written to the corresponding signals.

Composite Data Types

Another challenge in converting from SECoP to Ophyd is the difference in the supported data types that can be held in Signals and Parameters. Ophyd Signals are limited to scalars and arrays of the primitive data types *number*, *string* and *boolean*, as specified by the JSON specification [12], while SECoP has broader capabilities. Parameters can hold composite data types such as tuples and structs, which can be represented by mixed type JSON arrays and JSON ob-

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM010

jects. As composite datatypes cannot be represented by a single Signal, they are modelled recursively by sub-devices and Signals. Additionally, a Signal of type *string* is added. included, which contains the raw JSON object or array that was cast to a string. These string type Signals containing the raw JSON object or string are needed when a value needs to be written to such a Parameter. Writing to sub-device Signals would result in the *struct* or *tuple* being sent with only partially updated information.

Similar problems arise when representing arrays of composite data types. These arrays cannot be decomposed into subdevices and signals, and thus, they are represented as string arrays with the JSON value of the composite data type cast to string as entries.

TESTING WITH ACTUAL HARDWARE

The integration of SECoP-Ophyd was successfully tested on a Universal Robots UR3 robot. The SEC node that directly interfaced with the robot is composed of several Readable and Drivable modules and is designed to use the robot as a sample changer. After demonstrating the ability to execute Bluesky plans on the Ophyd devices generated by the SECoP-Ophyd integration in isolation, a heterogeneous setup was also tested. This involved the concurrent operation and integration into Bluesky plans alongside an ophyd.v1 device that represents a Keysight multimeter via EPICS process variables.

CONCLCUSION AND FUTURE PLANS

The integration of SECoP into Ophyd that we presented in this paper is an important step towards promoting the easy integration of sample environment equipment into beamlines at neutron and photon research facilities, that use Bluesky for experiment specification and orchestration. It has been demonstrated that both hardware abstractions utilise comparable concepts, which can be converted into each other without significant loss. Structural and functional representations of a SEC node and its associated modules, properties and parameters have been formalised, and appropriate ways of expressing the functionality of readable and drivable interface classes in Ophyd have been demonstrated. Consequently, we demonstrate the ability to collect and integrate sample environment metadata provided by a SEC node with metadata from the control system. Furthermore, the use of the Ophyd.v2 API has enabled the representation of SECoP commands within Ophyd. A suitable software architecture was found to implement the SECoP-Ophyd integration incorporating a SECoP-client as backend, ensuring a clear separation of concerns and promoting code reuse.

As a next step, optional features supported by Bluesky should be investigated to determine if SECoP can provide information and features to support them in a standardized and formal manner. Moreover, compatibility of Ophyd devices generated by the SECoP-Ophyd integration with the Nexus Format [13] requires further investigation. Lastly it is needed to apply the concept to real use cases at beamlines where sample environment equipment has to be integrated into an existing Bluesky environment, while further development of the SECoP-Ophyd integration is carried out on a GitHub repository [14] of the International Society for Sample Environment.

ACKNOWLEDGMENT

The authors would like to thank all those involved in the SECoP@HMC (ZT-I-PF-3-040) project, past and and present. The project is supported by the Helmholtz Metadata Collaboration (HMC), an incubator-platform of the Helmholtz Association within the framework of the Information and Data Science strategic initiative. Funding for the project was issued by the Initiative and Networking Fund of the Helmholtz Association in the framework of the HMC project call.

REFERENCES

[1] K. Kiefer *et al.*, "An introduction to SECoP – the sample environment communication protocol", *J. Neutron Res.*, vol. 21, no. 3-4, pp. 181–195, 2020. doi:10.3233/jnr-190143

[2] ISSE, https://sampleenvironment.org/

[3] ophyd, https://github.com/bluesky/ophyd

[4] D. Allan, T. Caswell, S. Campbell, and M. Rakitin, "Bluesky's ahead: A multi-facility collaboration for an a la carte software project for data acquisition and management", *Synchrotron Radiat. News*, vol. 32, no. 3, pp. 19–22, 2019. doi:10.1080/08940886.2019.1608121

[5] SECoP, https://github.com/SampleEnvironment/SECoP

[6] yaqc-bluesky, https://github.com/bluesky/yaqc-bluesky

[7] K. F. Sunden, D. D. Kohler, K. A. Meyer, P. L. C. Parrilla, J. C. Wright, and B. J. Thompson, "The yaq project: Standardized software enabling flexible instrumentation", *Rev. Sci. Instrum.*, vol. 94, no. 4, p. 044 707, 2023. doi:10.1063/5.0135255

[8] ophyd-tango, https://github.com/bluesky/ophyd-tango

[9] pycertifspec, https://github.com/SEBv15/pycertifspec

[10] Frappy, https://github.com/SampleEnvironment/frappy

[11] E. Gamma, Ed., *Design patterns, Elements of reusable object-oriented software*. Addison-Wesley, 2011, 395 pp.

[12] D. Crockford and C. Morningstar, "Standard ECMA-404 The JSON Data Interchange Syntax", Rep. ECMA 404, 2017. doi:10.13140/RG.2.2.28181.14560

[13] M. Könnecke *et al.*, "The NeXus data format", *J. Appl. Crystallogr.*, vol. 48, no. 1, pp. 301–305, 2015. doi:10.1107/s1600576714027575

[14] secop-ophyd, https://github.com/SampleEnvironment/secop-ophyd