Journal of Neutron Research 21 (2019) 181–195
DOI 10.3233/JNR-190143
IOS Press

An introduction to SECoP – the sample environment communication protocol

Klaus Kiefer [a,*], Lutz Rossa [a], Frank Wutzler [a], Niklas Ekström [b], Anders Pettersson [b], Enrico Faulhaber [c] and Markus Zolliker [d]

[a] *Helmholtz-Zentrum Berlin für Materialien und Energie, Hahn-Meitner-Platz 1, D-14109 Berlin, Germany*
E-mails: klaus.kiefer@helmholtz-berlin.de, rossa@helmholtz-berlin.de, frank.wutzler@helmholtz-berlin.de
[b] *European Spallation Source ERIC, SE-224 84 Lund, Sweden*
E-mails: niklas.ekstrom@esss.se, Anders.Pettersson@esss.se
[c] *Technische Universität München, Heinz Maier-Leibnitz Zentrum, D-85747 Garching, Germany*
E-mail: Enrico.Faulhaber@frm2.tum.de
[d] *Paul Scherrer Institut, CH-5232 Villigen, Switzerland*
E-mail: markus.zolliker@psi.ch

Abstract. The Sample Environment Communication Protocol (SECoP) serves as an international standard for the communication between sample environment equipment and the experiment control software at neutron and photon sources. It eases the integration of sample environment equipment supplied by external research groups and by industrial manufacturers. SECoP is designed to be simple, inclusive and self-describing. SECoP facilitates and structures the provision of metadata which is associated with sample environment equipment. Several existing implementations of SECoP support the development of SECoP-compatible sample environment control software. This article introduces SECoP Version 1.0, the first official version of SECoP published in September 2019. It was developed during the SINE2020 project in close cooperation with the International Society for Sample Environment. The complete specifications of SECoP Version 1.0 are available on GitHub.

Keywords: Sample environment, software, neutron, photon

1. Motivation

The Sample Environment Communication Protocol (SECoP) is intended to facilitate the integration of sample environment equipment (cryostats, humidity cells, magnets, etc.) into beamline experiments at neutron and photon sources. SECoP defines a standard for the communication between the software (or firmware) that is controlling the Sample Environment Equipment (SEE) and the beamline Experiment Control System (ECS). In addition, SECoP incorporates a standardized way to automatically provide metadata information about sample environment equipment. The SECoP logo is shown in Fig. 1.

Fig. 1. The SECoP logo.

*Corresponding author. E-mail: klaus.kiefer@helmholtz-berlin.de.

Viewed from the SEE control software or Programmable Logic Controller (PLC), SECoP is the top layer for the communication with the external world (such as an ECS). Hence, SECoP does not define how the sample environment equipment is controlled internally by the SEE control software or PLC. SECoP only defines the exchange layer for the information needed to control sample environment equipment by the ECS during a beamline experiment. In SECoP, the functionality of a sample environment device is represented by a Sample Environment Control Node (SEC-node). This SEC-node is the bridge from the SEE control software or PLC to the ECS.

SECoP was developed, defined and tested by the SECoP working group of SINE2020 WP 7.1 (HZB, ESS, MLZ, PSI) in close collaboration with the Committee for the Standardization of Sample Environment Communication of the International Society for Sample Environment (ISSE) [1]. During the SINE2020 project different (beta) versions of SECoP were defined. With the completion of the SINE2020 project, the first official version SECoP Version 1.0 (V2019-09-16 v1.0) is published. The complete specifications of SECoP Version 1.0 are available on the SECoP GitHub web site [2].

2. SECoP philosophy

SECoP is designed with an underlying philosophy. The most prominent principles for SECoP are that it should be:

- simple,
- inclusive,
- self-describing.

Simple means that SECoP must be easy to implement and to use. On the one hand the simplicity of SECoP is needed for its implementation in SEE control performed with hardware of potentially limited capacity like PLCs or microcontrollers. On the other hand, SECoP should also be simple enough to allow its implementation by non-expert programmers. This need for simplicity, however, does not compromise the use of SECoP for complex sample environment equipment.

Inclusive means, that SECoP must allow for different design concepts of the ECS and SEE control software (e.g. synchronous vs. asynchronous communication). Facilities can use SECoP without having to change their work flow (rewrite drivers completely or organize and handle hardware in a specific way to fulfil SECoP requirements).

Self-describing means that with SECoP, not only the pure data is transported. It also transports metadata, which allows the ECS to configure itself for communication with the SECoP SEE. The main driving force behind the development of SECoP was to reduce the effort needed to integrate new sample environment equipment. Connecting sample environment equipment to an ECS for a beamline experiment should be possible in a plug-and-play approach. With SECoP being self-describing, the description of a SEC-node must contain all necessary information for operating the SEC-node by the ECS without further documentation in, at least, a basic mode. It must also provide all relevant metadata information.

In addition, the following principles are part of the underlying SECoP philosophy:

- Definitions must be necessary, sufficient and unambiguous. Don't define what does not have to be defined.
- The transport layer must be byte oriented (TCP/IP, serial).
- The protocol must be independent from the specific transport layer.
- Complex functionality of the sample environment equipment (on the SEC-node side) must be wrapped so that a simplified and standardized use of SE equipment by the ECS is possible.
- Basic SECoP plug-and-play operation must always be possible.
- Keep the overhead for the SECoP on the SEC-node (server) side small (for the use in e.g. PLCs).
- Avoid unnecessary traffic.
- Better to be explicit.
- All protocol messages must be human readable (with the only exception: data type `blob`).
- Names in SECoP are treated as case sensitive but must be unique if lowercased.

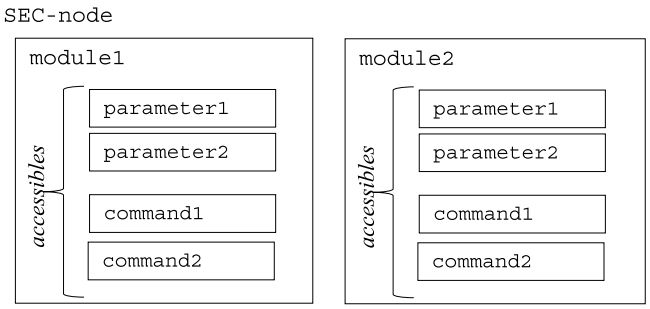

Fig. 2. Schematic illustration of the SECoP structure.

- Use JSON.
- Impose best-practices to the programmer of the SEC-node by making essential features mandatory.
- Apply must-ignore policy (to allow further extensions of the protocol in a compatible way).
- Allow for multiple clients.
- If multiple clients are connected to a SEC-node, only one of them should change parameters or send commands. Otherwise resulting problems might not be handled by SECoP.
- There should be a general way of doing things (exceptions must be motivated very well).

3. Definition of SECoP version 1.0

The SECoP Version 1.0 (V2019-09-16 v1.0) was released on 16 September 2019 by the SINE2020 SECoP working group and the Committee for the Standardization of Sample Environment Communication of the ISSE. This section gives an overview of:

- the general structure of the SECoP information,
- the syntax for the SECoP messages,
- the set of rules for the SECoP logic,
- the meaning of predefined SECoP components.

The complete specification of SECoP Version 1.0 is available on GitHub [2].

3.1. SECoP basic structure/hardware abstraction

The basic structure of SECoP is illustrated by a Sample Environment Control Node (SEC-node), see Fig. 2. A SEC-node represents the server side (the SE hardware side) of a SECoP connection, and its building blocks are modules. Modules represent the physical quantities of a sample environment device (for example the temperature of a cryostat or the field in a magnet). Each module can have accessibles (parameters and commands), which give additional control or live information about the modules. Static information about the SEC-node, modules and accessibles are in turn retained by their respective properties.

Examples for properties are the `description` property (for SEC-node, module and accessibles) or the `datainfo` property for accessibles. The `datainfo` property contains all relevant information about the (mandatory) data type, the (optional) data range and the (optional) physical unit of the accessible. For more detailed information see the SECoP GitHub site [2].

3.2. SECoP syntax

The complete SECoP syntax cannot be described in this short introduction. A complete description is available online on the SECoP GitHub web site [2]. However, the general principles are presented here.

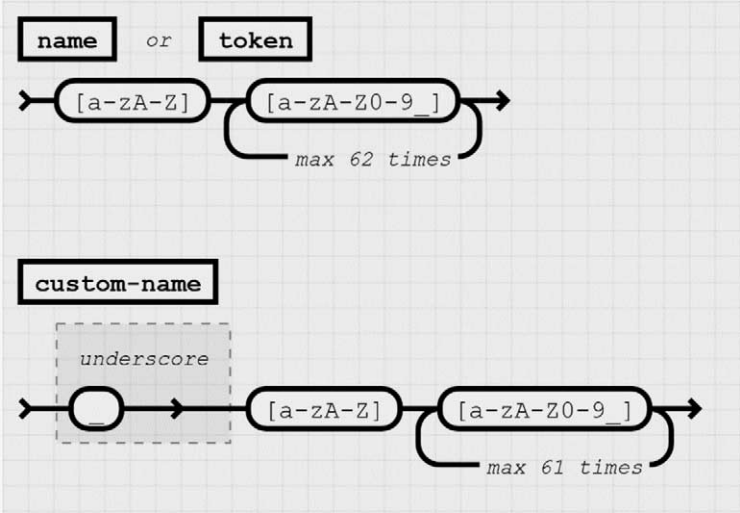

Fig. 3. Allowed SECoP identifiers.

3.2.1. SECoP identifiers

All identifiers in SECoP (for example names for properties, accessibles and modules) are composed of ASCII letters, digits and underscore, where a digit MUST NOT appear as the first character. The maximum length of an identifier is 63 characters. The rules for generating SECoP identifiers are depicted in Fig. 3.

Identifiers for custom accessibles, custom properties or custom messages which are not predefined in SECoP MUST start with an underscore ('custom-names'). Custom accessibles and in particular custom parameters can be used without limitations. However, the use of custom properties and custom messages is restricted to special tasks like debugging, where the ECS has full information about the meaning and especially the data type of the custom elements. Custom properties and custom messages which are unknown to an ECS will be ignored. Therefore, custom properties and custom messages should be avoided in order to ensure full compatibility between different implementations of SECoP.

Albeit names MUST be compared and stored case sensitive, names in each scope MUST also be unique when lowercased (to avoid problems for ECSs that are not case sensitive). The scopes are:

- module names on a SEC Node (including the group entries of those modules),
- accessible names of a module (including the group entries of those parameters or commands, each module having its own scope),
- properties (SEC-nodes, modules and accessibles have their own scope),
- names of elements in a struct (each struct has its own scope, for details on the data type struct see Section 4.10 in [2]),
- names of variants in an enum (each enum has its own scope, for details on the data type enum see Section 4.5 in [2]),
- names of qualifiers (for details on qualifiers see Section 3.2.3b).

SECoP defined names are usually lowercase, though that is not a restriction (especially not for module names).

3.2.2. SECoP messages

The basic element of SECoP are messages that are exchanged between the ECS and the SEC-node. Messages are delimited by a line feed character (LF, ASCII 10). All messages share the same basic structure: the message starts with an action keyword, followed optionally by one space and a specifier (not containing spaces), followed

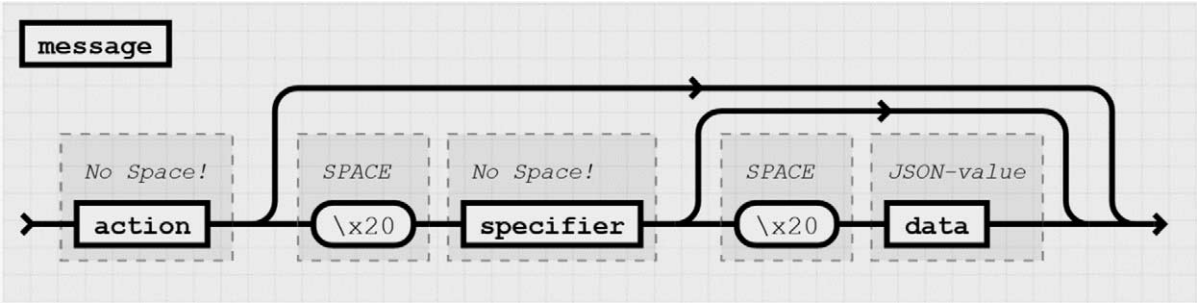

Fig. 4. General structure of a SECoP message.

optionally by one space and a JSON-value (see RFC 8259) called data, which absorbs the remaining characters up to the final LF (see Fig. 4).

The specifier is normally the name of a module followed by a ":" and the name of an accessible. In some cases (e.g. the `ping` message), it can be just a token i.e. an identifier.

Examples for basic SECoP messages (that must be implemented by all SECoP devices) are the `describe`, `read` and `change` messages. For a full list see [2]. The syntax for the mentioned messages is:

```
describe
read␣<module>:<parameter>
change␣<module>:<parameter>␣<Value>
```

`<module>`, `<parameter>` and `<Value>` have to be replaced by the actual module name, parameter name and the appropriate value. `<Value>` is, as stated in Fig. 4, a JSON-value. For the messages `read` and `change`, the actual strings sent to a SEC-node containing a module named `temp1` with the parameters `value` and `target` are (`target` is set to 295 K):

```
read temp1:value
change temp1:target 295
```

3.2.3. SECoP reply messages

All messages sent by the SECoP client to the SECoP server are prompted by a reply. The following examples illustrate the syntax of the reply messages to the request messages already presented in Section 3.2.2.

a) The "describing" message

The syntax for the reply to a `describe` message is:

```
describing␣.␣<Structure Report>
```

Comment: the dot (second item in the `describing` message) is a placeholder for extensibility reasons. An ECS client implementing the current specification MUST ignore the dot, while a SEC-node MUST generate it.

The `<Structure Report>` is a structured JSON describing the structure of the complete SEC-node. This includes the SEC-node properties, the modules, the module properties, accessibles and the properties of the accessibles. For details see [2].

For the example given in Section 3.2.2 with the SEC-node just containing one module (temperature) with the two parameters `value` and `target` and the additional parameter `status`, the `describing` message could have the following form (note: the original message is just one line, line feeds and indentation are added here for better readability):

describing.

```
{
 "description":"TestNode",
 "equipment_id":"HZB_Testnode-1",
 "firmware":"Isaac Version 1.7.0 with SHALL server library
             (Git 70591a14f66f37b92dcf6386a17159b526fa2913)",
 "modules":
 {
  "temp1":
  {
   "interface_classes":["Writable","Readable"],
   "description":"test module for a temperature controller",
   "accessibles":
   {
    "value":
    {
     "description":"temperature",
     "datainfo":{"type":"double","unit":"K"},
     "readonly":true
    },
    "target":
    {
     "description":"target temperature",
     "datainfo":{"type":"double","unit":"K","min":0,"max":300},
     "readonly":false
    },
    "status":
    {
     "description":"module status",
     "datainfo":
     {
      "type":"tuple",
      "members":
      [
       {
        "type":"enum",
        "members":{"DISABLED":0,"IDLE":100,
                   "WARN":200,"BUSY":300,
                   "STABILIZING":380,"ERROR":400}
       },
       {
        "type":"string"
       }
      ]
     },
     "readonly":true
    }
   }
  }
```

```
  }
}
```

Please note that this module is a representative of the basic SECoP interface class `Writable` which extends upon `Readable`. More information on interface classes can be found in [2].

b) Replies to "read" and "change" messages

The syntax for the replies to the `read` and `change` messages is:

```
reply␣<module>:<parameter>␣<Data Report>
changed␣<module>:<parameter>␣<Data Report>
```

The `<Data Report>` of the `reply` and `changed` messages is a JSON array with the value of a parameter as its first element (for complex data types this will be a complex JSON-value), and a JSON-object containing the qualifiers for this value as its second element. Qualifiers optionally augment the value in a reply from the SEC-node, and present variable information about that parameter. They are collected as named values in a JSON-object. Presently SECoP V1.0 defines two qualifiers:

"t": UNIX like time stamp with fractional seconds, i.e. seconds since 1970-01-01T00:00:00+00:00Z
"e": the uncertainty of the quantity. MUST be in the same units as the value.

The answers to the examples for the `read` and `change` messages of Section 3.2.2 could be:

```
reply temp1:value [295.13,{"t":1505396348.188}]
changed temp1:target [300,{"t":1505396349.123}]
```

3.2.4. Error reply

The SEC-node indicates by an error reply that the requested action could not be performed as requested (by the ECS) or that other problems occurred. The error reply consists of

1. the action keyword "error_" plus the original action keyword of the original request,
2. the original specifier of the request (if any),
3. the error-report: a JSON-array containing the name of one of the error classes (a string from a list of predefined errors classes, see [2]), an error message (a human readable string specifying the actual problem) and as a third element a JSON-object containing extra error information, which may include the timestamp (as key "t") and possible additional implementation specific information about the error (stack dump etc.),

separated by single space characters. Examples for error replies are listed below (">" indicates the request by the ECS, "<" indicates the reply by the SEC-node):

```
> read temp2:value
< error_read temp2:value ["NoSuchModule","temp2 is not configured on this
SEC node", {}]

> change temp1:ramp 1
< error_change temp1:ramp ["NoSuchParameter","temp1 has no parameter ramp",
{}]

> change temp1:target -9
< error_change temp1:target ["BadValue","requested value (-9) is outside
limits (0..300)", {}]

> reaaad temp1:target
< error_reaaad temp1:target ["ProtocolError","unknown action", {}]
```

Fig. 5. Syntax of the error-report (created by a SEC-node).

The syntax for the error report is depicted in Fig. 5.

3.2.5. Asynchronous messages

SECoP supports synchronous and asynchronous communication. A SEC-node MUST support both and a SECoP client (ECS) may use either or both. With synchronous communication the SEC-node only sends information if requested by the ECS. In asynchronous mode the SEC-node will send `update` messages for parameters that have changed. The asynchronous mode is activated by the `activate` message sent by the ECS to a SEC-node. For details see [2].

The syntax for the `update` message is:

```
update␣<module>:<parameter>␣<Data Report>
```

This definition is similar to the `reply` message presented in Section 3.2.3. An example for an `update` message for the parameter `value` of the module `temp1` could be:

```
update temp1:value [294.93,{"t":1505396354.232}]
```

3.2.6. Commands

In SECoP, commands are provided to initiate specified actions of a module. They should generate an appropriate reply immediately after that action is initiated, i.e. should not wait until some other state is reached. However, if the command triggers side-effects, they MUST be communicated to all clients that are in asynchronous mode BEFORE the reply is sent. If an action needs significant time to complete (i.e. longer than a fraction of a second), the information about the duration and success of such an action has to be transferred via the status parameter.

Commands are defined in a similar way as parameters are with their `type` information in the `datainfo` property defined as `command` (see Data info in [2]). Commands may use a single, possibly structured argument and may return a single, possibly structured result. Commands with a predefined meaning are listed in the SECoP standard, they must always be used in the same way.

The following commands and their meaning are predefined in SECoP V1.0 (for details see section “Accessibles” in [2]):

- `stop`
- `communicate`
- `reset`
- `clear_error`
- `go`
- `hold`
- `shutdown`

A command is executed by the `do` message. The following example shows the use of the `stop` command:

```
> do temp1:stop
< done temp1:stop [null,{"t":1505396348.876}]
```

Comment: The `stop` command has no argument, so the data part of the `do` message is omitted here. The `stop` command has no return value, hence the `null` in the data report.

3.2.7. Identification

The first messages to be exchanged after a new connection between an ECS and a SEC node is established, are to verify that indeed the SEC node is speaking a supported protocol. This is done by the ECS by sending an identification request and checking the answer from the SEC node to comply. If this check fails, the connection is to be closed and an error reported.

The syntax of the identification message `*IDN?` differs a little bit from other messages, as it should be compatible with IEEE 488.2. The reply consists of 4 comma separated fields, where the second and third field determine the used protocol. For example:

```
> *IDN?
< ISSE&SINE2020,SECoP,V2019-09-16,v1.0
```

3.3. SECoP rules

SECoP relies on the strict following of some basic rules when implementing a SEC-node or an ECS (or other SECoP client). In this section the most important rules for SECoP are described.

3.3.1. Connections

SECoP is not restricted to a transmission form like TCP/IP over Ethernet or a serial connection over RS232. However, the implementations presently available all realize SECoP over TCP/IP.

For all connections the following rules apply:

- After the establishment of a connection, the SECoP client (ECS) has to send an identification message (`*IDN?`) to the SEC-node checking the compatibility of the SECoP version. If positive, a `describe` request must follow. The answer provides the client with all available static information about the SEC-node.
- Both, the SECoP Server (SEC-node) and SECoP Client (ECS) must handle an interruption of the connection at any time.
- The SEC-node may close a connection that is considered inactive at any time. Hint: the `ping` request issued by the ECS (see "heartbeat" in [2]) may help to keep a connection open.
- After a temporary interruption of the connection: the ECS has to apply the same procedure as with a new connection.
- If the configuration of a SEC-node is changed in a way that influences the content of the `describing` message, the SEC-node has to close the connection. The client detects this and has to reopen the connection.
- The SEC-node should support multiple client connections. The SEC-node is, however, not responsible for the mess multiple clients can possibly create if not synchronized.
- SECoP does not handle security of transferred data nor access control and relies on support by other means.

3.3.2. Handshake

In SECoP, a well-defined handshake procedure ensures that the ECS is always able to know, when a request and the connected actions are finished. The handshake procedure includes two elements:

- the request/reply pair,
- the IDLE → BUSY → IDLE sequence.

Every request of the ECS is answered by a reply of the SEC-node. When the expected reply is received, the ECS knows that all actions connected with the request were at least started. However, the reply can as well be an error message informing the ECS about the non-performance.

The IDLE → BUSY → IDLE sequence informs the ECS about the accomplishment of all actions that were connected with the request (e.g. a temperature change). Actions that take considerable time have to set the predefined

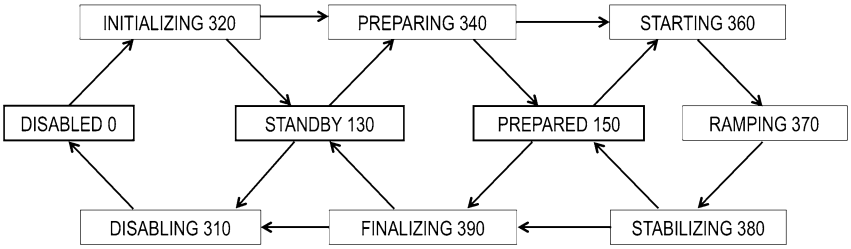

Fig. 6. Sketch of the allowed transitions between status codes. Please notice: not all of the mentioned status codes have to appear during the progress of a process. However, the sketch is indicating the allowed order of the appearance of the status codes. (This issue is still partially under discussion).

parameter `status` of a module to a BUSY state (for more information on `status` see [2]). When all actions are finished, the `status` parameter is set to an IDLE state.

If the `status` parameter is not defined in a module, this module is not allowed to deal with actions that take considerable time. The `status` parameter is mandatory for the interface classes `Readable`, `Writable` and `Drivable`.

The combination of both elements, the request/reply pair and the IDLE → BUSY → IDLE sequence, form the SECoP handshake procedure. When the reply was received and the `status` parameter (that was read after the reply message) is IDLE, the ECS knows that all necessary actions to complete the request are accomplished.

The following rules apply for the SECoP handshake:

- On the SEC-node side, all side effects of a request have to be dealt with before any reply is sent to the ECS.

This is especially true for setting the status to BUSY if an action takes considerable time and for sending update messages for all parameters before sending the `active` reply after an `activate` message (asynchronous mode).

Hint: The side effects can be effective in modules other than the module for which the request was received.

- On the SEC-node side, actions that take considerable time to be finished (like changing a magnetic field or driving a motor) must not delay the reply message. This behaviour must be dealt with by setting the `status` parameter to BUSY before sending the reply message.
- The SEC-node may reject almost every request at any time by sending an error message. In particular, this is true if the `status` parameter is in a BUSY state. The only exception is the predefined `stop` command which must always be processed.

3.3.3. Flow control

The flow control enables the ECS to keep track of the progress of its requests. The flow control is realized by the different status codes of the `status` parameter of a module.

Whereas the first digit of the status code is relevant for the handshake (for example all status codes starting with 1 (e.g. 130) represent IDLE states and all status codes starting with 3 (e.g. 370) represent BUSY states), the second digit of the status code is reserved for the flow control of a generic state machine (for details see `status` in [2] and Fig. 6).

The following rule applies for the SECoP flow control:

- Use the predefined `status` parameter and the predefined status codes of the generic state machine for flow control.

3.3.4. Plug & play

The plug & play functionality of SECoP can only be effective if some rules are obeyed:

- All SECoP servers (SEC-nodes) and clients (e.g. ECSs) must implement a must-ignore policy to ensure compatibility with coming SECoP versions and with custom solutions. For example, unknown properties or qualifiers must be ignored by the ECS.

- Use the timestamp and uncertainty qualifiers whenever possible.
- Use the predefined parameters. Don't invent new custom parameters if avoidable.
- Use the predefined properties of SEC-node, modules and accessibles. Always give meaningful descriptions, units, limits of the range of values, etc. when possible.
- Modules must always be downwards compatible with the basic interface classes if applicable. For modules dealing with physical properties the interface classes `Readable`, `Writable` and `Drivable` can and must be used.

3.3.5. Additional rules

Some rules do not fit in the above categories but still have to be applied for SECoP compatibility:

- All SEC-nodes have to offer both synchronous and asynchronous communication (see "activate updates" in [2]).
- In asynchronous mode, the SEC-node must send update messages to all clients that have activated the asynchronous mode (see activate updates in [2]).
- A `read` request on a parameter should be answered with a reasonably fresh value. What "reasonable" means depends on the specific application. In some cases, a stored value is the only way of supplying a value in a short time, especially if it would take too long to read out a fresh value from the (slow) hardware.
- SECoP supports "live" and "buffered" modes. In live mode, all `change` requests are performed immediately, whereas in buffered mode you need to send a `go` command. The presence of the `go` command decides which mode the implementer chose for the specific module. For example, setting the `target` parameter of a module will directly issue the movement only in the case that no `go` command is defined in the module. If the `go` command is defined, the movement will only start after sending the `go` command. This can be used for a buffering mechanism where e.g. the `target_ramp` and other parameters have to be set before starting the change.

3.4. Meaning of predefined SECoP components

The SECoP syntax was explained in Section 3.2. With this information an ECS can for example read or control all accessibles.

An integral part of SECoP is the predefined meaning of SECoP components. With the knowledge of the predefined meaning of for example a SECoP parameter, the ECS will be able to automatically interpret the given information or save the information in a sorted way. For this purpose, the SECoP standard includes the possibility to transport the meaning for selected properties, modules and accessibles with:

- the basic predefined properties,
- the optional module property `meaning`,
- the interface classes.

3.4.1. Predefined properties

The basic predefined properties are essential for the basic plug & play functionality of SECoP. The definition of a property contains:

- the name,
- the data type,
- the meaning,
- possible proposals for the structure of the contained information (see for example the SEC-node property `equipment_id`).

For a full list of predefined properties, see [2].

3.4.2. The property "meaning"

The optional module property `meaning` publishes the signification of a module in the context of the experiment in form of keyword and an attributed number (`importance`). Examples for valid keywords are `temperature` (representing the sample temperature), `humidity` or `magneticfield`. The list of predefined keywords [2] will be extended if needed. The numerical attribute `importance` defines the priority of the assignment in the case that different modules with the same meaning keyword are present in one SEC-node.

With the knowledge of the `meaning`, an ECS can interpret the functionality of a module in the correct way. In the context of metadata (see Section 4), the knowledge of the `meaning` enables the ECS to save the metadata in an appropriate way.

3.4.3. Interface classes

The interface classes define collections of accessibles in a module with predefined functionality and meaning. Besides the standard interface classes (see [2]), interface classes for e.g. magnetic field or temperature controllers are in the process of being defined. As an example, the module in the `describing` message in section SECoP Reply Messages supports the standard interface class `Readable` with the predefined parameters `value` and `status` and the derived standard interface class `Writable` with the additional predefined parameter `target`.

In the context of metadata (see Section 4), the interface classes give the possibility of a predefinition of the meaning of specific parameters and thus enable the automatic attribution of metadata information, for example the calibration curve of a temperature sensor.

The definition of complex interface classes will be an ongoing task for the further development of SECoP.

4. Metadata

The need to provide a suitable and complete set of metadata for beamline experiments was one of the main motivations for the development of SECoP. The task of providing metadata involves two different aspects: on the one hand metadata has to be provided in a standardized format to be automatically processed by the ECS. On the other hand, the meaning of the offered metadata has to be well defined in order to ensure a correct future interpretation and evaluation of the datasets. Both aspects are covered by the definition of SECoP. The meaning of predefined SECoP components was already presented in Section 3.4.

In SECoP, metadata can be divided into two major categories: live and static. Live metadata can change during a measurement. Static metadata remains unchanged during a measurement.

4.1. Live metadata

Live metadata consists of all metadata information that can be altered during the experiment. For example, supporting sample environment data like the filling level of a helium cryostat or the water bath temperature of a humidity cell will change during the time of an experiment and are therefore referred as live metadata. The knowledge of the live metadata can be essential for the interpretation of the experimental results, especially when it has some influence on the major sample environment variables like the sample temperature or the humidity at the sample position. In the context of some ECSs, even the major sample environment variables like sample temperature or the magnetic field at the sample position might be considered as live metadata.

In SECoP, there is no formal discrimination between live metadata and other forms of live data. Consequently, live metadata is provided as the value of single parameters and updated in the standard way by synchronous (`read`) or asynchronous (`update`) messages. The discrimination between live metadata and other live data is purely done by the attributed meaning (see Section 3) and the interpretation by the ECS.

4.2. Static metadata

Static metadata must not and cannot change during a running experiment. SECoP provides all static metadata only as an answer to a `describe` message. In the case that static metadata has changed due to a reconfiguration of the sample environment equipment, the complete SEC-node has to be disconnected from all clients and consecutively reconnected again. Examples for static metadata are the description of a SEC-node or the calibration curve of a sensor. In SECoP, the description of a SEC-node is provided as a predefined property of the SEC-node. The calibration curve of a sensor however is a complex data structure. SECoP provides static parameters for this kind of numerical or structured metadata.

Of course, the structure of a SEC-node with its modules and their accessibles can also be seen as metadata. This structure is natively contained in the answer to a `describe` message.

5. Implementations of SECoP

Parallel testing of the current SECoP version through integration in both sample environment control software as well as in experiment control software have been an integral part of the definition process of SECoP. The partners of the SECoP workgroup (HZB, ESS, PSI, MLZ) developed independent realizations of SECoP which were tested against each other to verify correct function of the syntax. Each implementation has a slightly different focus and has been developed in different environments. This procedure was intended to test the completeness and the unambiguousness of the definition of SECoP.

5.1. Support for the integration of SECoP

Following the underlying philosophy of SECoP, SECoP is intended to be simple and easy to be implemented. However, the complete integration of SECoP from scratch into a sample environment controller and in particular into an experiment control software (ECS) can be a time-consuming process – all depending on the complexity of the sample environment equipment or control software. In addition, it has to be kept in mind that some control software or firmware for sample environment equipment will be developed by non-IT experts.

For the purpose of an easy integration of SECoP into existing or future applications, three SECoP implementations were developed during the SINE2020 project:

- the **SHALL libraries**, which can be called from high-level programming languages with C-compatible interface, implementing SECoP clients and servers,
- the Python based **Frappy framework** for programming sample environment control nodes with a SECoP interface,
- the **EPICS focused framework (work name: octopus)**, embedding SECoP in the EPICS world.

They provide SECoP libraries or frameworks developed for different environments and different programming languages.

The integration of SECoP via the use of the SHALL libraries, the Frappy framework or the EPICS focussed framework offers the implementer an easy way to integrate correct SECoP functionality. In addition, it makes the software future-proof as most changes of possible future new versions of SECoP will be processed within the libraries and frameworks. The SHALL libraries, the Frappy framework and the octopus EPICS focused framework developed into software packages are or will be made publicly available [3–5].

5.2. First integrations into existing experiment control systems

During the development phase, SECoP was integrated in different experiment control systems:

- CARESS (HZB)

- SICS/SEA (PSI)
- NICOS2 (MLZ)

In CARESS, due to internal design concepts, only the basic functionality of SECoP can be accessed. However, CARESS offers already the full integration of the predefined basic interface classes Readable, Writable and Drivable. Thus, the plug-and-play concept of SECoP could successfully be adopted in CARESS. As a consequence, at HZB, sample environment equipment with SECoP functionality can already be integrated into neutron scattering experiments.

For SICS, the current ECS at the spallation neutron source of PSI, a SECoP client was developed, passing through the sample environment component SEA. This covers most of the SECoP functionality.

The current integration of SECoP into NICOS2 needs to be adapted to the latest changes in the protocol. A former version was tested with real hardware via the MLZ tango-entangle interface layer. SECoP integrated nicely into the existing NICOS2 ECS, but the event handling implementation of the client needs a further iteration as NICOS2 relies on its own polling. The integration of SECoP in NICOS2 provided by MLZ will furthermore facilitate testing of SECoP systems at ESS as the ESS ECS construction is reaching completion.

In addition, a test implementation of SECoP was integrated successfully into NOMAD (ILL) to control sample environment equipment from MLZ at ILL.

5.3. First integrations into existing sample environment control systems

During the development of SECoP, numerous test integrations of SECoP into experiment control software were realized. Two sample environment systems were physically shipped to other facilities testing the functionality with SECoP implementations from different facilities.

- The oven system “stressihtf2” from MLZ was used and successfully controlled at ILL (with a basic SECoP client programmed by J. Locatelli, ILL). The control hardware of the system consists of a Eurotherm temperature controller and a Beckhoff PLC. The SECoP SEC-node offered control over the temperature, valves and monitored the cooling water.
- The SECoP control for the resistive magnet “Garfield” from MLZ was implemented. The control hardware of the system consists of a Lambda power supply and a Beckhoff PLC. The SECoP SEC-node offered control over the magnetic field at the sample position and monitored e.g. the coil temperature (metadata). The system was used at PSI and controlled by a SEA/SICS client programmed by M. Zolliker.

The integration of SECoP into sample environment control software via the SHALL Server Library was successfully tested with different sample environment equipment at HZB (e.g. temperature controllers, needle valve controllers) both for LabVIEW and Delphi (“Isaac” control system) based software. The integration proved to be simple and quick. Similarly, several sample environment apparatuses at MLZ were equipped with SECoP, allowing to control the hardware via the (already full integrated) specific entangle-servers. Control was re-routed in NICOS2 to use SECoP instead of the native entangle client. As SECoP was mainly used as a bridge (with the full functionality still in the entangle-servers), integration was very easy. The whole system ran smoothly and reliably.

6. Summary

The publication of SECoP Version 1.0 marks a milestone in the long process of standardization of sample environment equipment for large-scale research facilities. All major neutron sources in Europe were involved in the development of SECoP. In addition, with the participation of the ISSE in the process of the definition, SECoP aims to be a worldwide standard for sample environment control at neutron and photon sources.

The definition of SECoP includes the structures and methods to provide and to communicate metadata for sample environment related applications. The predefined meaning that is included in SECoP allows for automated

and well sorted storage of metadata. The mapping to the NeXus data format is generally straightforward and can be automatized.

First realizations of SECoP are already implemented. With the publication of SECoP Version 1.0 and the public availability of the SHALL Libraries as well as the Frappy and EPICS focused frameworks, a fast increase of the number of SECoP implementations is expected in the very near future.

The core team from HZB, ESS, MLZ and PSI will continue to lead the development of SECoP after the end of the SINE2020 project. This, and the use of SECoP in all facilities represented by the core team, will provide a good foundation for the future development of SECoP.

Acknowledgements


We thank very much the numerous colleagues from the research facilities not belonging to the core group for their valuable inputs. The work described was performed as part of the world class Science and Innovation with Neutrons in Europe 2020 (“SINE2020”) project, which was funded from the European Union’s Horizon 2020 research and innovation programme under grant agreement No.°654000. The syntax diagrams were generated using a modified copy of https://github.com/EnricoFaulhaber/railroad_dsl.

References

[1] https://sampleenvironment.org.
[2] https://github.com/sampleenvironment/SECoP.
[3] https://github.com/sampleenvironment/SHALL.
[4] https://github.com/sampleenvironment/Frappy.
[5] https://github.com/sampleenvironment/octopus.