Open Geospatial Consortium Inc.

Date: 2010-06-30

Reference number of this document: OGC 10-134

Category: OpenGIS® Discussion Paper

Editors: Arne Broering, Stefan Below

**OpenGIS® Sensor Interface Descriptors**

Copyright © 2010 Open Geospatial Consortium, Inc.
To obtain additional rights of use, visit [http://www.opengeospatial.org/legal/](http://www.opengeospatial.org/legal/).

**Warning**

This document is not an OGC Standard. This document is an OGC Discussion Paper and is therefore not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, an OGC Discussion Paper should not be referenced as required or mandatory technology in procurements.

Document type: OpenGIS® Discussion Paper
Document subtype: N.A.
Document stage: Approved for public release
Document language: English
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Scope</td>
<td>1</td>
</tr>
<tr>
<td>2  Normative references</td>
<td>1</td>
</tr>
<tr>
<td>3  Terms and definitions</td>
<td>2</td>
</tr>
<tr>
<td>4  Conventions</td>
<td>2</td>
</tr>
<tr>
<td>4.1 Abbreviated terms</td>
<td>2</td>
</tr>
<tr>
<td>4.2 UML notation</td>
<td>3</td>
</tr>
<tr>
<td>5  Introduction to Sensor Web Enablement</td>
<td>3</td>
</tr>
<tr>
<td>6  The Gap between Sensor Web and Sensor Layer</td>
<td>4</td>
</tr>
<tr>
<td>7  Sensor Interface Descriptors</td>
<td>6</td>
</tr>
<tr>
<td>7.1 Overview on SID Utilization</td>
<td>6</td>
</tr>
<tr>
<td>7.2 Encapsulation of the SID</td>
<td>7</td>
</tr>
<tr>
<td>7.3 Definition of Data Flow</td>
<td>7</td>
</tr>
<tr>
<td>7.4 Definition of Addressing Parameters</td>
<td>9</td>
</tr>
<tr>
<td>7.5 Definition of Sensor Protocol</td>
<td>9</td>
</tr>
<tr>
<td>7.6 Definition of Protocol Processing</td>
<td>12</td>
</tr>
<tr>
<td>7.6.1 Character Escaping Process Type</td>
<td>14</td>
</tr>
<tr>
<td>7.6.2 Checksum Process Type</td>
<td>15</td>
</tr>
<tr>
<td>7.6.3 Interpolation Process Type</td>
<td>17</td>
</tr>
<tr>
<td>7.6.4 Date Conversion Process Type</td>
<td>19</td>
</tr>
<tr>
<td>7.7 Definition of Observation Metadata</td>
<td>21</td>
</tr>
<tr>
<td>7.8 Definition of Sensor Commands</td>
<td>23</td>
</tr>
<tr>
<td>8  Interpreter Implementation</td>
<td>27</td>
</tr>
<tr>
<td>9  Conclusions &amp; Outlook</td>
<td>28</td>
</tr>
<tr>
<td>10 Acknowledgment</td>
<td>29</td>
</tr>
</tbody>
</table>

## Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1 - UML notations</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2 - Sensor Infrastructure Stack</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3 - Usage of SID in SWE deployment</td>
<td>6</td>
</tr>
<tr>
<td>Figure 4 - Overview of SID schema included in SensorML. Elements added by the SID schema are colored in blue.</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5 - Data flow between sensor and SWE through the SID</td>
<td>8</td>
</tr>
<tr>
<td>Figure 6 - SID extension of physical layer</td>
<td>10</td>
</tr>
<tr>
<td>Figure 7 - SID ComponentListType</td>
<td>10</td>
</tr>
</tbody>
</table>
Figure 8 - SID Encoder and Decoder of a *LayerPropertyType ............................................ 13
Figure 9 - SID CommandType .................................................................................................. 25
Figure 10 - Design of the SID Interpreter ............................................................................... 27
Figure 11 - Sensor Bus concept .............................................................................................. 28
Figure 12 - Draft of SID creator GUI ..................................................................................... 29

Tables

Table 1 — List of URNs for connection specification ............................................................... 9
Table 2 — Mandatory properties for character escaping process .......................................... 14
Table 3 — Parameters of checksum process ........................................................................... 15
Table 4 — Mandatory properties for checksum computation process .................................. 16
Table 5 — Mandatory properties for checksum validation process ...................................... 16
Table 6 — Mandatory properties for interpolation process .................................................... 18
Table 7 — Mandatory properties for date conversion process .............................................. 19
Table 8 — Literals for date conversion ..................................................................................... 20
Table 9 — List of URNs to characterize command parameters ............................................ 24
i. **Preface**

This document presents the Sensor Interface Descriptor (SID) schema that enables the declarative description of sensor interfaces, including the definition of the communication protocol, sensor commands, processing steps and metadata association. This schema is designed as a profile and extension of SensorML. Based on this schema, SID interpreters can be implemented, independently of particular sensor technology, which are able to translate between sensor protocol and SWE protocols. They establish the connection to a sensor and are able to communicate with it by using the sensor protocol definition of the SID. SID instances for particular sensor types can be reused in different scenarios and can be shared among user communities. The ability of an SID interpreter to connect sensors and Sensor Web services in an ad hoc manner based on the sensor’s SID instance is a next step towards realizing sensor plug & play within the Sensor Web.

ii. **Document terms and definitions**

This document uses the standard terms defined in Subclause 5.3 of [OGC 05-008], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this specification.

iii. **Submitting organizations**

The following organizations submitted this document to the Open Geospatial Consortium Inc.

*52° North Initiative for Geospatial Open Source Software GmbH*

*Institute for Geoinformatics (IfGI) of the University of Muenster*

iv. **Document contributor contact points**

All questions regarding this document should be directed to the editor or the contributors:

<table>
<thead>
<tr>
<th>Name</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stefan Below</td>
<td>IfGI</td>
</tr>
<tr>
<td>Arne Broering</td>
<td>52° North</td>
</tr>
<tr>
<td>Felix Bache</td>
<td>52° North</td>
</tr>
<tr>
<td>Thomas Everding</td>
<td>IfGI</td>
</tr>
</tbody>
</table>
v. Revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Editor</th>
<th>Primary clauses modified</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.03.2010</td>
<td>0.0.1</td>
<td>Broering</td>
<td>Throughout</td>
<td>Initial setup</td>
</tr>
<tr>
<td>02.06.2010</td>
<td>0.0.1</td>
<td>Broering</td>
<td>Throughout</td>
<td>Addition of examples and Annexes</td>
</tr>
<tr>
<td>05.06.2010</td>
<td>0.0.1</td>
<td>Broering</td>
<td>Throughout</td>
<td>Major polishing</td>
</tr>
<tr>
<td>08.06.2010</td>
<td>0.0.1</td>
<td>Everding</td>
<td>Throughout</td>
<td>Review and comments</td>
</tr>
<tr>
<td>09.06.2010</td>
<td>0.0.1</td>
<td>Bache</td>
<td>Outlook</td>
<td>Description of graphical SID creator added</td>
</tr>
<tr>
<td>09.06.2010</td>
<td>0.0.1</td>
<td>Broering</td>
<td>Throughout</td>
<td>Incorporation of comments</td>
</tr>
<tr>
<td>09.06.2010</td>
<td>0.0.1</td>
<td>Below</td>
<td>Throughout</td>
<td>Review and comments</td>
</tr>
</tbody>
</table>

vi. Changes to the OGC Abstract Specification

The OpenGIS® Abstract Specification does not require changes to accommodate the technical contents of this document.

vii. Future work

The developed approach has been tested under “lab conditions” with different sensor types. Next, it will be applied in real-world scenarios to demonstrate its benefits in sensor asset management. It is anticipated that the design will evolve as new issues and opportunities arise. The gained experimental feedback will be included in the next version of the SID schema.
Foreword

This document presents the concept of Sensor Interface Descriptors (SID). SID is a schema that enables the declarative description of sensor interfaces, including the definition of the communication protocol, sensor commands, processing steps and metadata association. The schema is designed as a profile and extension of the OGC SensorML (SML) schema.

This document includes 3 annexes; Annexes A and B are normative, and Annex C is informative.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.
Introduction

The Sensor Web Enablement (SWE) initiative of the Open Geospatial Consortium (OGC) defines standards for Web Service interfaces and data encodings that encapsulate sensors for web-based discovery, tasking and access. SWE has been applied in a multitude of projects, demonstrating its suitability in real world scenarios. However, there is still a fundamental challenge to be tackled. While SWE enables interoperability with the upper application layer, the connection between SWE and the underlying sensor layer and its heterogeneous protocols is not yet sufficiently described.

This work addresses this challenge and presents a declarative schema for Sensor Interface Descriptors (SID) based on OGC's SensorML standard. A SID for a particular sensor enables a SID interpreter to translate between the communication protocol of the sensor and the Sensor Web.

The SID concept is the basis for achieving the long-term vision of true sensor plug & play within the Sensor Web and will finally make sensors on-the-fly available on the Sensor Web.
OpenGIS® Sensor Interface Descriptors

1 Scope
This work presents the concept of Sensor Interface Descriptors (SID), a schema which enables the declarative description of sensor interfaces, including the definition of the communication protocol, sensor commands, processing steps and metadata association. This schema is designed as a profile and extension of SensorML. Based on this schema, SID interpreters can be built which are able to translate between sensor protocol and SWE protocols. Such interpreters can be built independently of particular sensor technology. They establish the connection to a sensor and are able to communicate with it by using the explicit sensor protocol definition of the SID. SID instances for particular sensor types can be reused in different scenarios and can be shared among user communities. The SID concept is a next step towards the long term vision of realizing sensor plug & play within the Sensor Web.

2 Normative references
The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

[OGC 07-000] OGC™ Sensor Model Language (SensorML) Standard, Version 1.0

[OGC 07-122r2] OGC™ SensorML Encoding Standard v 1.0 Schema Corregendum 1 (1.01), Version 1.0.1


In addition to this document, this specification includes several normative XML Schema Document files as specified in Annexes A and B.

### 3 Terms and definitions

For the purposes of this standard, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 05-008] and the SensorML Implementation Specification [OGC 07-000] shall apply.

### 4 Conventions

#### 4.1 Abbreviated terms

Some more frequently used abbreviated terms:

- **API**: Application Program Interface
- **GML**: Geography Markup Language
- **HTTP**: HyperText Transfer Protocol
- **ISO**: International Organization for Standardization
- **OGC**: Open Geospatial Consortium
- **OWS**: OGC Open Web Services
- **O&M**: Observations and Measurements
- **SAS**: Sensor Alert Service
- **SensorML**: Sensor Model Language
- **SES**: Sensor Event Service
- **SID**: Sensor Interface Descriptor
- **SOAP**: Simple Object Access Protocol
- **SOS**: Sensor Observation Service
- **SPS**: Sensor Planning Service
- **SWE**: Sensor Web Enablement
- **UML**: Unified Modeling Language
- **URI**: Uniform Resource Identifier
- **URL**: Uniform Resource Locator
- **URN**: Uniform Resource Name
4.2 UML notation

Some of the diagrams in this document are presented using the Unified Modeling Language (UML) static structure diagram. The UML notations used in this document are described in Figure 1, below.

**Figure 1 - UML notations**

In these UML class diagrams, the class boxes with a light background are the primary classes being shown in this diagram, often the classes from one UML package. The class boxes with a gray background are other classes used by these primary classes, usually classes from other packages.

5 Introduction to Sensor Web Enablement

The Sensor Web Enablement (SWE) [1] initiative of the Open Geospatial Consortium (OGC) defines standards which can be used as building blocks of the Sensor Web. The SWE framework decouples sensor data from the way they are collected and makes them available over the web through standardized formats and interfaces. The main Web Services of the SWE framework are the Sensor Observation Service (SOS), the Sensor Planning Service (SPS), the Sensor Alert Service (SAS), and the Sensor Event Service (SES). The SOS [2] is designed for accessing real time as well as historic sensor data, and
Sensor metadata descriptions. While the SOS follows the pull-based communication paradigm, SAS [3] and SES [4] are capable of pushing sensor data to subscribers. To control and task sensors the SPS [5] can be used. A common application of SPS is to define simple sensor parameters such as the sampling rate but also more complex tasks such as mission planning of satellite systems.

Apart from these Web Service specifications, SWE incorporates an information model for observed sensor data, the Observations & Measurements (O&M) [6] standard, as well as an information model for the description of sensors, the Sensor Model Language (SensorML) [7].

SensorML specifies a generic model and XML encoding for sensor related processes such as measurement procedures or post processing procedures. Physical as well as logical sensors are modeled as processes. The functional model of a process can be described in detail, including its identification, classification, inputs, outputs, parameters, as well as characteristics such as temporal availability or spatial description. Processes can be composed by process chains.

O&M defines a model and encoding for observations. An observation has a result (e.g. 0.7 mSv/a) which is an estimated value of an observed property (e.g. radiation), a particular characteristic of a feature of interest (e.g. the city center of Muenster). The result value is generated by a procedure, in general a sensor (e.g. a radiation detector) described in SensorML. These four central components are linked within SWE concepts.

5.1 The Gap between Sensor Web and Sensor Layer

In recent years, the SWE standards have been applied in various projects (e.g. [8, 9, 10, 11]) showing their practicability and suitability in real world scenarios. However, there is still an essential challenge to be tackled. As pointed out by [12], there is a gap of interoperability between the Sensor Web layer, the architectural level of the SWE services, and the sensor layer (see Figure 2). SWE defines service interfaces from an application oriented perspective. The connection of a SWE service to sensors is not sufficiently specified. Although, SAS and SOS allow the upload of sensor data, the utilization of the according operations still requires reformatting of the sensor protocol to the SWE protocol. A sensor itself is usually not able to upload its data directly to a SWE service, since its bandwidth and processing power are typically limited and the SWE protocol is rather complex and verbose. Most obvious is the interoperability gap at the SPS. It is not defined by the specification how an SPS transforms a retrieved sensor task to a command of the sensor protocol.
Today, the connection between sensors and SWE services is usually established by manually adapting the internals of the SWE service implementation to the specific sensor type. Such adaptations have to be built for each pair of service implementation and sensor type which leads to extensive efforts in developing large-scale sensor network systems [13].

5.2 Approach

This work addresses the identified interoperability gap with the concept of Sensor Interface Descriptors (SID). Chapter 6 presents a schema for SIDs which enables the declarative description of sensor interfaces, including the definition of the communication protocol, sensor commands, processing steps and metadata association. The schema is designed as a profile and extension of the SensorML standard. The SID schema is illustrated by examples which are excerpts of an SID instance for a radiation detector of the German Federal Office for Radiation Protection. The complete SensorML description of this radiation detector is presented in Annex C.

Based on this schema, SID interpreters can be built which are able to translate between sensor protocol and Sensor Web protocols and hence close the described interoperability gap. Such interpreters for SID instances can be built independently of particular sensor technology. They establish the connection to a sensor and are able to communicate with it by using the explicit sensor protocol definition of the SID. Chapter 7 outlines the implementation of a generic SID interpreter which has been developed in context of this work. It transfers data, retrieved from a sensor, to a Sensor Observation Service and transforms tasks, submitted to a Sensor Planning Service, to commands which are then forwarded to a sensor.

SID instances for particular sensor types can be reused in different scenarios and can be shared among user communities. The ability of an SID interpreter to connect sensors and Sensor Web services in an ad hoc manner based on the sensor’s SID instance is a next step towards realizing sensor plug & play within the Sensor Web.
6 Sensor Interface Descriptors

This Chapter presents the SID concept and schema. First, an overview on the usage of SIDs in a SWE deployment is given (Section 6.1). Next, the requirement for encapsulation of the SID within the SensorML document is emphasized (Section 6.2). The data flow between sensor and SWE service through the SID is described in Section 6.3. Then, the different aspects of the SID schema are described. Section 6.4 outlines how the basic addressing parameters of a physical connection to the sensor are specified. After establishing the physical connection, a definition of the raw sensor protocol is needed, which is illustrated in Section 6.5. With its definition, the sensor protocol can be interpreted and further processed, as described in Section 6.6. Before retrieved, interpreted and processed sensor data can be forwarded to SWE services, certain observation metadata has to be added, which is outlined in Section 6.7. To enable the tasking of sensors, the commands accepted by the sensor interface are described in Section 6.8.

6.1 Overview on SID Utilization

To give an overview of how to utilize an SID Interpreter and SID instances in real world scenarios, Figure 3 shows the deployment of a SWE infrastructure including the usage of SIDs. A sensor communicates with a data acquisition system in its specific sensor protocol over a transmission technology such as ISDN or GSM. This sensor can also act as a sensor gateway (or “network sink”) so that other nodes of a (possibly mobile) sensor network communicate with it. The SID interpreter runs on the data acquisition system and uses SID instances for the different sensors of the sensor network to translate between the sensor specific protocol and the SWE protocols. The interpreter is responsible to register a sensor at a SWE service and to upload sensor data to an SOS, SAS, or SES. Also, it is responsible for the opposite communication direction and forwards tasks received by an SPS to a sensor.

![Figure 3 - Usage of SID in SWE deployment](image-url)
6.2 Encapsulation of the SID

The SID is strictly encapsulated within the SensorML structure. This way, it can be easily replaced or removed. This characteristic is important since the SensorML document has to be usable in situations where the detailed interface description is not of interest or shall not be publicly accessible. Further, an encapsulation of the SID makes it possible to reuse it in different use cases for sensors with same interfaces. The ability of reusing SIDs also enables the installation of repositories to share SIDs for particular sensor types among user communities.

For these reasons, the approach developed here, encapsulates all SID specific information within the *sml:interface* element of an *sml:System*. The *interface* element contains a stack of layers (Figure 4), aligned with the Open System Interconnection (OSI) reference model [14]. In contrast to the OSI model, SensorML does not further define how to use these layers. The SID schema makes use of this layer stack and concretes its usage to describe the sensor interface.

If a sensor has more than one interface (e.g. an Ethernet interface for the data output and an RS-232 interface for sensor programming) the SensorML document can list more than one interfaces containing separate SIDs.

![Figure 4](image)

**Figure 4** - Overview of SID schema included in SensorML. Elements added by the SID schema are colored in blue.

6.3 Definition of Data Flow

The description of the data flow between the components of the different layers is essential for the SID schema. Figure 5 illustrates the data flow between sensor and SWE through the SensorML document and the SID. To reflect this data flow within the SID, the *sml:connections* element, which is also associated with the *sml:System*, is reused. The *sml:connections* element is associated with the *InterfaceDefinition* (Figure 4). This
allows the description of the data flow within the SID internally, and ensures the required encapsulation of the SID.


**Figure 5 - Data flow between sensor and SWE through the SID**

The arrows in Figure 5 symbolize the data flow through the SID. They can be represented by `sml:Link` elements in the `sml:connections` element associated with the `InterfaceDefinition`. Listing 1 shows two examples of such SID internal links. This first link defines a data output component of the `physicalLayer` (the time tag of the incoming sensor data) as the input of a decoder component (a date conversion process) located on the `networkLayer`. The second link points the output of the decoder process to a data output component on the `presentationLayer`. In the SID external `sml:connections` element of the `sml:System`, this output component of the `presentationLayer` is then linked to an output of the sensor system.

**Listing 1 - Example of SID internal connection**

```xml
<xsml:connections>
  <xsml:ConnectionList>
    <xsml:connection>
      <xsml:Link type="urn:ogc:def:link:OGC:cid"/>
      <xsml:source ref="physicalLayer/dataOutputComponents/M01/data/timeTag"/>
      <xsml:destination ref="networkLayer/decoderComponent/inputs/date01"/>
    </xsml:Link>
    <xsml:connection>
      <xsml:Link type="urn:ogc:def:link:OGC:cid"/>
      <xsml:source ref="networkLayer/decoderComponent/outputs/date01"/>
      <xsml:destination ref="presentationLayer/dataOutputComponents/odl_basis_01/samplingTime"/>
    </xsml:Link>
  </xsml:ConnectionList>
</xsml:connections>
```
6.4 Definition of Addressing Parameters

The addressing parameters (e.g. port and baud rate of a serial connection) are the basis for establishing a physical connection to the sensor. This physical connection is established through the operating system which runs the SID interpreter. The addressing parameters are stored externally in a document accompanying the SID, since the SID can be published publicly (e.g. via a SWE service) and the addressing parameters are security relevant.

Listing 2 - Definition of addressing parameters

```
<sml:interface name="mws3" xlink:role="urn:ogc:def:connection:OGC:serial">
   <sml:InterfaceDefinition>

The `xlink:role` attribute of the `sml:interface` is used to specify the type of the connection. Details of the connection for a particular interface, which is identified by the `name` attribute, are stated in an external file whose structure is SID interpreter dependent. The type of connection is specified by using a Unified Resource Name (URN) pointing to globally defined semantics. The currently allowed URNs and their definitions are listed in Table 1.

Table 1 — List of URNs for connection specification

<table>
<thead>
<tr>
<th>Names</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:ogc:def:connection:OGC:serial</td>
<td>Serial connection</td>
</tr>
<tr>
<td>urn:ogc:def:connection:OGC:tcpIP</td>
<td>TCP/IP connection</td>
</tr>
<tr>
<td>urn:ogc:def:connection:OGC:http</td>
<td>HTTP connection</td>
</tr>
<tr>
<td>urn:ogc:def:connection:OGC:udpIP</td>
<td>UDP connection</td>
</tr>
<tr>
<td>urn:ogc:def:connection:OGC:file</td>
<td>File system</td>
</tr>
</tbody>
</table>

The usage of the `xlink:role` element to define the connection is formally specified as Schematron rules (see Annex B.5). Those rules which are part of the SID schema can be considered as a profile of the SensorML schema.

6.5 Definition of Sensor Protocol

For the declarative definition of a sensor protocol, the exact definition of the raw data streams exchanged between sensor and data acquisition system is essential. The structure of this raw data is described within the `physicalLayer` element. As shown in Figure 4 and Figure 6 new elements for the data input and data output stream are attached to the element. The two elements are necessary to support duplex communication with sensors.
Figure 6 - SID extension of physical layer

Figure 7 shows the structure of the sid:ComponentListType which is used to model the incoming or outgoing data stream. The data can be described by using the sml:IOComponentType or the swe:DataBlockDefinition. The latter one also allows the specification of an encoding.

Listing 3 - Example data output stream coming from an MWS3 sensor

Station|1275482685|33UUU 932 592|10530Q|#
Status|1275482686|2|43|72|0|#
M01|1275482698|147.0|150.0|23.0|16.3|#
:

Listing 4 describes the input and output of an MWS3. The DataRecord element of the component named ‘M01’ defines the structure and meaning of each token of a single data block within the incoming data stream of measurement data. Data blocks are separated by the ‘#’ sign as defined in the encoding element. The field of the data record which is annotated with the URN urn:ogc:def:encoding:OGC:assertedValue identifies the data.

1 http://de.wikipedia.org/wiki/MWS3-Messwertsender
block of the data stream to which the structure definition refers. In the example of Listing 4, the data record of the component named ‘M01’ defines the structure for those data blocks where the first token has the value ‘M01’.

The component of the DataInputStream named ‘samplingrateCommand’ represents the data structure which is sent to an MWS3 sensor as a command to set its sampling rate. The detailed structure of commands is intentionally not defined on the physicalLayer but instead on the applicationLayer (Section 6.8). Only on the applicationLayer, a command consists of several subcomponents. On the layers below the applicationLayer, the command is considered as one aggregated data set. The reason for this is that the processes (e.g. character escaping or checksum validation processes), which are defined on the layers below the applicationLayer, need to be executed on the command as one data set.

The component of Listing 4 named ‘ackResponse’ defines the data structure returned by an MWS3 sensor as an acknowledgement response to the command which sets the sensor’s sampling rate.

Listing 4 - Description of incoming and outgoing MWS3 data stream

```xml
<sml:physicalLayer>
  <!-- incoming data stream: -->
  <sid:DataInputStream>
    <dataInputComponents>
      <ComponentList>
        <component name="samplingrateCommand">
          <swe:DataRecord>
            <swe:field name="command"/>
          </swe:DataRecord>
        </component>
      </ComponentList>
    </dataInputComponents>
  </sid:DataInputStream>

  <!-- outgoing data stream: -->
  <sid:DataOutputStream>
    <dataOutputComponents>
      <ComponentList>
        <!-- responses to sensor commands: -->
        <component name="ackResponse">
          <swe:DataBlockDefinition>
            <swe:components name="data">
              <swe:DataRecord>
                <swe:field xlink:role="urn:ogc:def:encoding:OGC:assertedValue" name="ack" value="0x06"/>
              </swe:DataRecord>
            </swe:components>
            <swe:encoding>
              <swe:BinaryBlock byteEncoding="hex" byteOrder="bigEndian">
                <swe:member>
                  <swe:Block ref="data/ack" byteLength="1"/>
                </swe:member>
              </swe:BinaryBlock>
            </swe:encoding>
          </swe:DataBlockDefinition>
        </component>
      </ComponentList>
    </dataOutputComponents>
  </sid:DataOutputStream>
</sml:physicalLayer>
```
6.6 Definition of Protocol Processing

For enabling the definition of processing steps which are necessary to translate between the sensor protocol and the SWE protocol, the dataLinkLayer, networkLayer, transportLayer, and sessionLayer are utilized. To allow data processing in both directions, from sensor domain to SWE domain and the other way round, elements for data decoding and encoding are added to each of these layers. Figure 8 shows this extension. Instances of the Encoder and Decoder elements contain either an sml:ProcessModel or sml:ProcessChain to define the process. A process model can be used to describe a single non-physical process with its inputs, outputs, parameters and its computational method. A process chain can be used to represent a chain of multiple processes and to encapsulate them as one process.
Each of the four layers is optional in an SID. Which processes are described on which layer, depends on the design of a particular SID instance. An interpreter executes the processes defined in these layers sequentially.

Due to their significance for sensor communication, an SID interpreter shall natively support the following four process types: (i) character escaping, (ii) checksum computation and validation, (iii) value interpolation, and (iv) date conversion.

For each of those process types a URN is defined, which can be specified in the method property of the process model. Besides those four natively supported processes, other process methods can be incorporated by describing them inline\(^3\) using Content MathML\(^4\).

An example for a typical usage of these four processes to encode a sensor data stream to higher level measurements and the association of the processes to layers of the SID can look like this:

1. `dataLinkLayer` specifies a process for character escaping.
2. `networkLayer` computes a checksum validation.
3. `transportLayer` transforms raw data by applying an interpolation.
4. `sessionLayer` computes a date conversion.

SensorML profiles for each of the four natively supported process types are described in the next subsections.

\(^2\) Note: This figure shows the NetworkLayerPropertyType. DataLinkLayerPropertyType, TransportLayerPropertyType and SessionLayerPropertyType are structured in the same way.

\(^3\) Note: The possibility of referencing binary or source code representations of processes is intentionally not supported by the SID schema, since such kinds of process definitions would depend on implementation of a particular SID interpreter.

\(^4\) http://www.w3.org/TR/MathML2
6.6.1 Character Escaping Process Type

In sensor communication, escape characters are used to induce an alternative interpretation of a transmitted character. As seen in the example of Listing 3, the end token of a data set within a data stream is indicated by a particular control character, the ‘#’ sign. In the raw sensor data, this control character is masked by an escape character (e.g. \). Which characters are used for escaping is defined by the sensor protocol.

Hence, every SID interpreter shall support a process type for removing and adding escape characters. The process type can be referenced by defining the URN urn:ogc:def:process:OGC:escCharacter in the method property. The mandatory properties of the character escaping process type are listed in Table 2. A formal definition of the character escaping process type as Schematron rules can be found in Annex B.1. Listing 5 shows an example usage of the character escaping process.

Table 2 — Mandatory properties for character escaping process type

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sml:inputs</td>
<td>Input data on which character escaping shall be performed.</td>
</tr>
<tr>
<td>sml:outputs</td>
<td>Processed output data.</td>
</tr>
<tr>
<td>sml:parameters</td>
<td>Details on escape characters as 2-dimensional swe:DataArray.</td>
</tr>
</tbody>
</table>

Listing 5 - Example of the character escaping process

```xml
<sml:ProcessModel>
  <sml:inputs>
    <sml:InputList>
      <sml:input name="data"/>
    </sml:InputList>
  </sml:inputs>
  <sml:outputs>
    <sml:OutputList>
      <sml:output name="data"/>
    </sml:OutputList>
  </sml:outputs>
  <sml:parameters>
    <sml:ParameterList>
      <sml:parameter name="escapeDefinition">
        <swe:DataArray>
          <swe:elementType name="escapeCharacters">
            <swe:DataRecord>
              <swe:field name="charFrom" xlink:role="urn:ogc:def:hex"/>
              <swe:field name="charTo"/>
            </swe:DataRecord>
          </swe:elementType>
          <swe:encoding>
            <swe:BinaryBlock byteEncoding="hex" byteOrder="littleEndian">
              <swe:member>
                <swe:Block ref="escapeCharacters/charFrom" byteLength="1"/>
              </swe:member>
              <swe:member>
                <swe:Block ref="escapeCharacters/charTo" byteLength="2"/>
              </swe:member>
            </swe:BinaryBlock>
          </swe:encoding>
        </swe:elementType>
      </sml:parameter>
    </sml:ParameterList>
  </sml:parameters>
</sml:ProcessModel>
```
6.6.2 Checksum Process Type

For a reliable sensor communication, the ability to compute and validate checksums is essential. Each SID interpreter shall offer a process type for that purpose. It can be referenced using the URN urn:ogc:def:process:OGC:1.0:checkSum.

The most widely used checksum method is the Cyclic Redundancy Check (CRC). However, there is no standard describing how to compute a CRC. Hence, the SID schema supports the parameterized model for the definition of CRC algorithms, the Rocksoft Model [15]. The parameters of this model (e.g. polynomial, and name of the algorithm to be used) are passed along in the parameters element of this process type. The necessary parameters and their meaning are described in Table 3.

The mandatory properties of the process type to compute a checksum are listed in Table 4. Listing 6 shows an example usage of the checksum computation process.

The mandatory properties of the process type to validate a checksum are listed in Table 5. For validating a checksum, the asserted value of a checksum shall be defined in the sml:inputs element of the process.

A formal definition of the checksum process type using Schematron rules can be found in Annex B.2.

Table 3 — Parameters of checksum process type
<table>
<thead>
<tr>
<th>Names</th>
<th>SWE Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Text</td>
<td>Name of algorithm to be used.</td>
</tr>
<tr>
<td>width</td>
<td>Count</td>
<td>Length of polynomial.</td>
</tr>
<tr>
<td>poly</td>
<td>Text</td>
<td>Specification of the polynomial in hex. The first bit is omitted.</td>
</tr>
<tr>
<td>init</td>
<td>Count</td>
<td>Initial value of the register at the beginning of the calculation.</td>
</tr>
<tr>
<td>refin</td>
<td>Boolean</td>
<td>Condition which has to be fulfilled before the execution can be started.</td>
</tr>
<tr>
<td>refout</td>
<td>Boolean</td>
<td>Indicates whether the register is forwarded directly to the XOROUT operation or whether a binary swap is invoked first.</td>
</tr>
<tr>
<td>xorout</td>
<td>Text</td>
<td>Hexadecimal value associated with the result of the XOR operation.</td>
</tr>
<tr>
<td>check</td>
<td>Text</td>
<td>Validation value of the algorithm for the String “123456789”</td>
</tr>
</tbody>
</table>

Table 4 — Mandatory properties for checksum computation process type

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sml:inputs</td>
<td>Data for which the checksum shall be computed.</td>
</tr>
<tr>
<td>sml:outputs</td>
<td>Computed checksum.</td>
</tr>
<tr>
<td>sml:parameters</td>
<td>See Table 3.</td>
</tr>
</tbody>
</table>

Table 5 — Mandatory properties for checksum validation process type

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sml:inputs</td>
<td>1. data - Data for which the checksum shall be computed. 2. checksum - asserted checksum for given data.</td>
</tr>
<tr>
<td>sml:outputs</td>
<td>Returns the input data if the validation is successful.</td>
</tr>
<tr>
<td>sml:parameters</td>
<td>See Table 3.</td>
</tr>
</tbody>
</table>

Listing 6 - Example of the checksum computation process

```xml
<sml:ProcessModel>
  <sml:inputs>
    <sml:InputList>
      <sml:input name="data"/>
    </sml:InputList>
  </sml:inputs>
  <sml:outputs>
    <sml:OutputList>
      <sml:output name="checkSum">
        <swe:Text/>
      </sml:output>
    </sml:OutputList>
  </sml:outputs>
  <sml:parameters>
    <sml:ParameterList>
      <sml:Parameter/>
    </sml:ParameterList>
  </sml:parameters>
</sml:ProcessModel>
```

¹ Note: A detailed description of these parameters can be found in [15].
6.6.3 Interpolation Process Type

Interpolations need to be computed to transform raw sensor data to observations, to compute calibrations, or to correct measurements. For example, an electric current returned by a detector needs to be transformed to an actual measurement value of a particular phenomenon (e.g. radiation or temperature).
This process type can be referenced by urn:ogc:def:process:OGC:interpolation:cubic or urn:ogc:def:process:OGC:interpolation:linear, depending on the interpolation method which shall be applied. The mandatory properties of this process type are listed in Table 6. A formal definition of this process type using Schematron rules can be found in Annex B.3. Listing 7 shows an example usage of the interpolation process.

Parameters of this process type are the two axes x and y between which the interpolation shall be performed. The actual values of the coordinate tuples defining the curve are usually defined outside of the SID. This is done, since the definition of the interpolation curve is depending on the sensor deployment, not the sensor interface. For example, a weather station might use different curve definitions to transform raw data to observations depending on the set up how and where it is installed. Listing 8 shows an example of an sml:Component which can be listed in the sml:components element of an sml:System and is hence outside of the SID. This sml:Component contains the parameterization of a curve for the two axes ‘resistance’ and ‘radiation’. The fields representing these two axes can be linked to the two axes of the interpolation process by specifying according links in the sml:connections element of the sml:System (see Annex C). This way, the interpolation process is externally parameterized.

Table 6 — Mandatory properties for interpolation process type

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sml:inputs</td>
<td>Value which shall be interpolated.</td>
</tr>
<tr>
<td>sml:outputs</td>
<td>Interpolated value.</td>
</tr>
<tr>
<td>sml:parameters</td>
<td>The two axes between which the interpolation shall be performed. The actual coordinate tuples defining the curve are usually defined outside of the SID.</td>
</tr>
</tbody>
</table>

Listing 7 - Example of a date interpolation process

```xml
<sml:ProcessModel>
  <sml:inputs>
    <sml:InputList>
      <sml:input name="value">
        <swe:Quantity/>
      </sml:input>
    </sml:InputList>
  </sml:inputs>
  <sml:outputs>
    <sml:OutputList>
      <sml:output name="result">
        <swe:Quantity/>
      </sml:output>
    </sml:OutputList>
  </sml:outputs>
  <sml:parameters>
    <sml:ParameterList>
      <sml:parameter name="settings">
        <swe:DataArray>
          <swe:elementCount/>
          <swe:elementType name="values">
            <swe:DataRecord>
              <swe:field name="x"/>
            </swe:DataRecord>
          </swe:elementType>
        </swe:DataArray>
      </sml:parameter>
    </sml:ParameterList>
  </sml:parameters>
</sml:ProcessModel>
```
Listing 8 - Example parameterization of a date interpolation process

```xml
<sml:component name="radiometer">
  <sml:Component>
    <sml:parameters>
      <sml:ParameterList>
        <sml:parameter name="steadyStateCurve">
          <swe:Curve>
            <swe:elementCount>
              <swe:Count><swe:value>2</swe:value></swe:Count>
            </swe:elementCount>
            <swe:elementType>
              <swe:SimpleDataRecord>
                <swe:field name="resistance">
                </swe:field>
                <swe:field name="radiation">
                </swe:field>
              </swe:SimpleDataRecord>
            </swe:elementType>
            <swe:encoding>
              <swe:TextBlock tokenSeparator = "," decimalSeparator = ",." blockSeparator = "; " />
            </swe:encoding>
            <swe:values>-3,-5; 6,7; 8.2,7.5</swe:values>
          </swe:Curve>
        </sml:parameter>
      </sml:ParameterList>
    </sml:parameters>
  </sml:Component>
</sml:component>
```
6.6.4 Date Conversion Process Type

If sensors tag their data with a timestamp, usually a conversion of the sensor time (e.g. milliseconds since Unix Epoch) to another time representation (e.g. ISO 8601) is necessary. This date conversion shall be natively supported by every SID interpreter. In the parameters element of this process type, literals are used to define the input and output time formats (e.g. a 'T' for Unix time in seconds, and 'YYYY' for the years with 4 digits). By specifying the URN urn:ogc:def:process:OGC:dateConversion it is indicated to the SID interpreter that this process method shall be applied. The mandatory properties of this process type are listed in Table 7 and the literals usable in date formatting are listed in Table 8. A formal definition of this process using Schematron rules can be found in Annex B.4. Listing 9 shows an example usage of the date conversion process.

Table 7 — Mandatory properties for the date conversion process type

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sml:inputs</td>
<td>Date string which shall be converted.</td>
</tr>
<tr>
<td>sml:outputs</td>
<td>Converted date string.</td>
</tr>
<tr>
<td>sml:parameters</td>
<td>Definition of date format of input and output using the literals defined in Table 8.</td>
</tr>
</tbody>
</table>

Table 8 — Literals for date conversion⁶

---

⁶ Note: This list of literals for the definition of time strings is influenced by the Java SimpleDateFormat (http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html)
<table>
<thead>
<tr>
<th>Literal</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Epoch (e.g. ‘AD’)</td>
</tr>
<tr>
<td>Y</td>
<td>Year</td>
</tr>
<tr>
<td>M</td>
<td>Month (e.g. ‘01’ = January)</td>
</tr>
<tr>
<td>w</td>
<td>Week of year</td>
</tr>
<tr>
<td>W</td>
<td>Week of month</td>
</tr>
<tr>
<td>D</td>
<td>Day of year</td>
</tr>
<tr>
<td>d</td>
<td>Day of month</td>
</tr>
<tr>
<td>F</td>
<td>Day of week</td>
</tr>
<tr>
<td>E</td>
<td>Day textual (e.g. ‘Wednesday’)</td>
</tr>
<tr>
<td>a</td>
<td>am/pm</td>
</tr>
<tr>
<td>H</td>
<td>Hour (0-23)</td>
</tr>
<tr>
<td>k</td>
<td>Hour (1-24)</td>
</tr>
<tr>
<td>K</td>
<td>Hour am/pm (0-11)</td>
</tr>
<tr>
<td>h</td>
<td>Hour am/pm (1-12)</td>
</tr>
<tr>
<td>m</td>
<td>Minutes of hour</td>
</tr>
<tr>
<td>s</td>
<td>Seconds of minute</td>
</tr>
<tr>
<td>S</td>
<td>Milliseconds of second</td>
</tr>
<tr>
<td>z</td>
<td>Time zone textual (e.g. ‘GMT’)</td>
</tr>
<tr>
<td>Z</td>
<td>Time zone as defined in RFC 822 (e.g. ‘-0800’)</td>
</tr>
<tr>
<td>T</td>
<td>Unix timestamp in seconds (number of seconds elapsed since midnight UTC of January 1, 1970)</td>
</tr>
<tr>
<td>t</td>
<td>Unix timestamp in milliseconds</td>
</tr>
</tbody>
</table>

Listing 9 - Example of a date conversion process

```xml
<sml:ProcessModel>
  <sml:inputs>
    <sml:InputList>
      <sml:input name="date">
        <swe:Text />
      </sml:input>
    </sml:InputList>
  </sml:inputs>
  <sml:outputs>
    <sml:OutputList>
      <sml:output name="date">
        <swe:Text />
      </sml:output>
    </sml:OutputList>
  </sml:outputs>
  <sml:parameters>
    <sml:ParameterList>
      <sml:parameter name="dateSettings">
        <swe:DataRecord>
          <swe:field name="inputFormat">
            <swe:Text>
              <swe:value>T</swe:value>
            </swe:Text>
          </swe:field>
        </swe:DataRecord>
      </sml:parameter>
    </sml:ParameterList>
  </sml:parameters>
</sml:ProcessModel>
```
6.7 Definition of Observation Metadata

The data, resulting from the preceding processing steps (Section 6.6), has to be associated with certain metadata, which is part of the O&M model, before it can be forwarded to an SOS. The measured data needs to be associated with units of measure so that an interpretation is possible. Further, the data needs to be linked to the elementary SWE components (Section 5), the observed property and the feature of interest, so that observations of the O&M model can be created and inserted into an SOS.

The association of the data with a unit of measure is done on the presentationLayer as shown in Listing 10. The DataOutputStream element is used to describe the data coming from the sensor on this layer. For example, the component named ‘odl_basis_01’ contains the field descriptions of the measured sensor data. The field named 'highDoseRadiation' represents the radiation data measured in millisievert per year ('mSv/a').

**Listing 10 - Definition of sensor data on presentation layer**

```xml
<sml:presentationLayer>
  <!-- incoming data stream: -->
  <sid:DataInputStream>
    <dataInputComponents>
      <ComponentList>
        <component name="samplingRateCommand">
          <swe:DataRecord>
            <swe:field name="command"/>
          </swe:DataRecord>
        </component>
      </ComponentList>
    </dataInputComponents>
  </sid:DataInputStream>

  <!-- outgoing data stream: -->
  <sid:DataOutputStream>
    <dataOutputComponents>
      <ComponentList>
        <component name="ackResponse">
          <swe:DataRecord>
            <swe:field name="ack"/>
          </swe:DataRecord>
        </component>
        <component name="odl_basis_01">
          <swe:DataRecord>
            <swe:field name="highDoseRadiation"/>
          </swe:DataRecord>
        </component>
      </ComponentList>
    </dataOutputComponents>
  </sid:DataOutputStream>
</sml:presentationLayer>
```
In the *sml:outputs* element of the *sml:System*, the sensor data is linked to the observed property, and to the feature of interest. Also, the data is linked to an observation offering, an element of the SOS which groups thematically similar observations. The linking to an observation offering is necessary for being able to execute the *InsertObservation* operation of the SOS. As shown in Listing 11, the *gml:metaDataProperty* is used for this linking. The feature of interest and the observed property are referenced by the means of the *xlink:href* attribute. For the observation offering, its service side identifier is given. The *field* element named ‘*dataSet*’ includes the description of the data by specifying the reference to the element with the *gml:id* ‘*highDoseRadiationData*’ (see Listing 10) in the *xlink:href* attribute.

The usage of those elements within the *sml:output* element to define the observation metadata is specified as Schematron rules (see Annex B.6). Those rules which are part of the SID schema can be considered as a profile of the SensorML schema.

The *sml:outputs* element is not part of the SID, since it is not a sub-element of the *InterfaceDefinition* (Figure 4). The information contained in the *sml:output* elements is intentionally kept out of the SID, due to the fact that the linkage of a sensor to a feature of interest, an observed property, and an observation offering is dependent on the particular use case, not the sensor interface. Not including this information into the SID, enables a reusing of the SID in different SWE deployments.

**Listing 11 - Definition of observation metadata in output element**

```xml
<sml:outputs>
  <sml:OutputList>
    <sml:output name="highDoseRadiation_output">
      <swe:DataRecord>
        <gml:metaDataProperty>
          <offering>RADIATION_OFFERING</offering>
        </gml:metaDataProperty>
      </swe:DataRecord>
    </sml:output>
  </sml:OutputList>
</sml:outputs>
```
6.8 Definition of Sensor Commands

The application layer of the OSI model describes interfaces to access the OSI stack. Compliant to this view, the **applicationLayer** is used here to define the commands accepted by the sensor. These command definitions can be used by an SPS so that it can provide information to clients how to task a sensor. Figure 9 shows the structure of the SID CommandType. The **command** element contains sub-elements to describe the responses of the sensor, the pre- and postconditions for executing the command, as well as the command parameters.

Each command has a mandatory attribute to define its unique **name**. The **timeout** attribute is of type **xs:integer** representing the time in milliseconds until a response of the sensor has to be received. The attribute **repeat** of type **xs:boolean** can be used to define whether a command call has to be repeated before a response is received. The attribute **auto** of type **xs:boolean** defines whether the command call is automatically executed every n seconds. The number of seconds is defined by the **interval** attribute.

The element **sid:CommandParameters** contains the parameters of the command. The **parameter** element is of type **sml:IoComponentPropertyType** and uses a **swe:DataRecord** to list the parameters in **swe:field** elements. The data type of the parameter can be specified using the basic SWE Common types (e.g. **swe:Text** or **swe:Count**). Additionally, the allowed value domain can be restricted using the **swe:constraint** element. The parameters can be further characterized using the URNs defined in Table 9 as value of the **xlink:role**. The order of the command parameters is the same as in the sensor protocol to which the command is mapped by the processes defined in the lower layers (Section 6.6).
Table 9 — List of URNs to characterize command parameters

<table>
<thead>
<tr>
<th>Names</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:ogc:def:command:OGC:name</td>
<td>Command name</td>
</tr>
<tr>
<td>urn:ogc:def:command:OGC:value</td>
<td>Constant value in the command definition (e.g. a separator sign)</td>
</tr>
<tr>
<td>urn:ogc:def:parameter:OGC:optional</td>
<td>Optional parameter</td>
</tr>
<tr>
<td>urn:ogc:def:parameter:OGC:required</td>
<td>Required parameter</td>
</tr>
</tbody>
</table>

Under consideration that a command might have different responses with different meanings, the element sid:ResponseList contains the possible responses of the command. The response element is of type sid:IoComponentType which extends the sml:IoComponentPropertyType by adding the valid attribute of type xs:boolean. This attribute marks whether the response represents a successful or unsuccessful execution of the command.

The element PostCondition shall be used to reference commands which are executed after execution of this command. The PreCondition element references commands which shall be executed before this command. The sub-elements assertTrue and assertFalse of PreCondition and PostCondition shall be used to demand a successful or unsuccessful execution of those referenced commands.
Figure 9 - SID CommandType
Listing 12 shows an example of a *CommandDefinition* of a command which sets the sampling rate of a sensor. The command has four parameters: the first one is the command name and fixed to the value 'SR', the second one is a text representing the sensor ID, the third one is a constant value ('|') acting as a separator sign, and the fourth one is the measuring interval, with a minimum value of 5 seconds.

**Listing 12 - Example of a command definition to set the sampling rate of a sensor**

```xml
<sml:applicationLayer>
  <sid:CommandDefinition>
    <sid:commands>
      <command name="setSamplingRate">
        <sid:CommandParameters name="setSamplingRateParameters">
          <swe:DataRecord>
            <swe:field name="cmd">
              <swe:Text xlink:role="urn:ogc:def:command:OGC:name">
                <swe:value>SR</swe:value>
              </swe:Text>
            </swe:field>
            <swe:field name="sensorID" xlink:role="urn:ogc:def:command:OGC:required">
              <swe:Text/>
            </swe:field>
            <swe:field name="separator" xlink:role="urn:ogc:def:command:OGC:value">
              <swe:Text>|</swe:Text>
            </swe:field>
            <swe:field name="interval" xlink:role="urn:ogc:def:command:OGC:optional">
              <swe:Quantity>
                <swe:uom code="sec"/>
                <swe:constraint>
                  <swe:AllowedValues>
                    <swe:min>5</swe:min>
                  </swe:AllowedValues>
                </swe:constraint>
              </swe:Quantity>
            </swe:field>
          </swe:DataRecord>
        </sid:CommandParameters>
        <sid:ResponseList>
          <response valid="true" name="successfulCommand">
            <swe:DataRecord>
              <swe:field name="ack"/>
            </swe:DataRecord>
          </response>
          <response valid="false" name="unsuccessfulCommand">
            <swe:DataRecord>
              <swe:field name="nack"/>
            </swe:DataRecord>
          </response>
        </sid:ResponseList>
        <PreCondition>
          <assertTrue ref="initCommand"/>
        </PreCondition>
      </command>
    </sid:CommandList>
  </sid:commands>
</sid:CommandDefinition>
</sml:applicationLayer>
```
7 Interpreter Implementation

This Chapter outlines a possible design of an SID interpreter\(^7\) as it has been developed by 52\(^\circ\) North. The implementation of the SID interpreter is based on the OSGi framework\(^8\) which is extendible by pluggable and loosely coupled Bundle components (Figure 10).

![Diagram of SID Interpreter](image)

**Figure 10 - Design of the SID Interpreter**

The implemented components of the 52\(^\circ\) North SID interpreter are the following:

- A central Manager component controls the workflow.
- The SID Parser is used to read in the SID document of the sensor.
- Depending on the specified addressing parameters (Section 6.4), a particular Data Source Connector implementation (e.g. for serial connections) is chosen to connect to the sensor.
- Based on the protocol definition of the SID (Section 6.5), the Protocol Transformer communicates with the sensor in a bi-directional way.
- The Process Executor is able to execute the four native process types (Section 6.6). Also, user-defined MathML processes can be executed by the means of the MathML Solver library\(^9\).
- The SOS Connector triggers the SOS operation RegisterSensor to add the new sensor to the Sensor Web and executes the InsertObservation operation to upload sensor data as observations (Section 6.7) to an SOS.
- The SPS Connector forwards the SensorML document and the contained SID to an SPS which uses the sensor command descriptions (Section 6.8) to provide detailed

---

\(^7\) [http://52north.org/sid](http://52north.org/sid)  
\(^8\) [http://www.osgi.org](http://www.osgi.org)  
\(^9\) [http://sourceforge.net/projects/mathmlsolver](http://sourceforge.net/projects/mathmlsolver)
information how to task the sensor. Sensor tasks, submitted to the SPS, are forwarded by the SPS to the SID interpreter and received by the *SPS Connector* component. The tasks are transformed to the sensor protocol, and passed through the *Data Source Connector* to the sensor.

### 8 Conclusions & Outlook

The concept of SIDs can minimize the efforts of integrating new sensors with the Sensor Web. Once created for a particular sensor type, an SID can be exchanged and reused in different contexts, projects, or user communities. In future, repositories can be set up which facilitate the sharing and the discovery of SID instances. By using an SID instance, the generic SID interpreter then allows an on-the-fly integration of sensors with the Sensor Web by minimizing the administration and deployment efforts. This is significant step towards the vision of sensor plug & play within the Sensor Web.

A further step towards vision will be the incorporation of the SID interpreter into the Sensor Bus\(^{10}\) architecture [16]. This publish/subscribe architecture (Figure 11) establishes an event-driven communication between SWE services and sensors. Thereby, a Sensor Adapter transforms the sensor protocol to the simple bus message protocol. It is planned to realize one generic Sensor Adapter by making use of the SID interpreter.

![Sensor Bus concept](http://52north.org/sensorBus)

---

\(^{10}\) http://52north.org/sensorBus
To support the creation of SIDs, the 52°North Sensor Web Community is currently developing tools and graphical user interfaces. This will lower the entry threshold for sensor network administrators to make their sensors available on the Sensor Web.

A first draft of a graphical user interface (GUI) which facilitates the creation of SDIs is shown in Figure 12. This GUI enables a user to create an SID by presenting separate sections for data input, processing and metadata association. Another version of this GUI contains a section for sensor tasking instead of the metadata association. This enables a user to model the data flow from an SPS to the sensor protocol. The data flow can be modeled intuitively by creating connections between the different components in a drag & drop manner.

The protocol section defines the structure of data coming from the sensor. New fields can be added and the format is defined. The user can drag the data fields to the processing section where new processes can be created with their according parameters. The processing outputs are then passed to the last section, where the metadata is added to the processed data.

9 Acknowledgment

This work is financially supported by the project "Flexible and Efficient Integration of Sensors and Sensor Web Services" funded by the European Regional Development Fund (ERDF) for NRW (grant number N 114/2008) of the European Union, as well as the 52° North Sensor Web community (http://52north.org/SensorWeb) which envisions a real time integration of heterogeneous sensors into a coherent information infrastructure.
Annex A
(normative)

XML Schema Documents

In addition to this document, this specification includes several normative XML Schema Documents. These XML Schema Documents are bundled in a zip file with the present document.

The XML Schema Documents developed for SIDs are based on the SensorML XML Schema Documents specified in [OGC 07-000]. However, for being able to extend SensorML in the way described in this specification, it needed to be slightly modified:

1. Each layer associated with the `sml:InterfaceDefinition` has gotten its own type instead of using `sml:LayerPropertyType` for all layers.
2. The `sml:connections` element has been associated with the `sml:InterfaceDefinition`.

The updated schema files are bundled with this document.

The abilities specified in this document use 4 specified XML Schema Documents included in the zip file with this document. These XML Schema Documents are named:

- basicElements.xsd
- command.xsd
- InterfaceDefinition.xsd
- sid.xsd
Annex B
(normative)

Schematron Rules

This Annex contains the Schematron definitions which act as a profile of the SensorML schema. First, the Schematron rules of the four processes natively supported by every SID interpreter are listed. Further, Schematron rules for the definition of addressing parameters and for the definition of observation metadata are listed.

B.1 Schematron rules of Character Escaping Process

```xml
<!DOCTYPE schema [
<!ENTITY input "sml:inputs/sml:InputList/sml:input">  
<!ENTITY output "sml:outputs/sml:OutputList/sml:output">  
<!ENTITY param "sml:parameters/sml:ParameterList/sml:parameter"> ]>
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
  </sch:title>Escape Character Process</sch:title>
</!-- Namespaces definitions -->
  <sch:ns prefix="sml" uri="http://www.opengis.net/sensorML/1.0.1"/>
  <sch:ns prefix="swe" uri="http://www.opengis.net/swe/1.0.1"/>
  <sch:ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/>

  <sch:pattern id="IO_CHECK" name="Check I/O Characteristics">
    </sch:rule context="$process;">
      <!-- Escape Character input -->
      <sch:assert test="$input[@name='data']">Input named 'data' must be present</sch:assert>
      <!-- Escape Character output -->
      <sch:assert test="$output[@name='data']">Output named 'data' must be present</sch:assert>
      <!-- Escape Character parameter -->
      <sch:assert test="$param[@name='escapeDefinition']">Parameter named 'escapeDefinition' must be present</sch:assert>
    </sch:rule>
  </sch:pattern>

  <sch:pattern name="CRC_Rocksoft">
    </sch:rule context="$param[@name='escapeDefinition']">
      <sch:assert test="swe:DataArray/swe:elementType[@name='escapeCharacters']/swe:DataRecord">
        DataRecord named 'escapeCharacters' must be present in escapeDefinition Block
      </sch:assert>
      <sch:assert test="swe:DataArray/swe:elementType[@name='escapeCharacters']/swe:DataRecord/swe:field[@name='charFrom']">
        Parameter named 'charFrom' must be present in escapeCharacters Block
      </sch:assert>
      <sch:assert test="swe:DataArray/swe:elementType[@name='escapeCharacters']/swe:DataRecord/swe:field[@name='charTo']">
        Parameter named 'charTo' must be present in escapeCharacters Block
      </sch:assert>
      <sch:assert test="swe:DataArray/swe:encoding">
        encoding must be present in escapeDefinition
      </sch:assert>
    </sch:rule>
```
B.2 Schematron rules of Checksum Validation Process

```xml
<!DOCTYPE schema [ 
<!ENTITY input "sml:inputs/sml:InputList/sml:input"> 
<!ENTITY output "sml:outputs/sml:OutputList/sml:output"> 
<!ENTITY param "sml:parameters/sml:ParameterList/sml:parameter"> ]> 
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron"> 
<sch:title>Checksum Process</sch:title> 
<!-- Namespaces definitions --> 
<sch:ns prefix="sml" uri="http://www.opengis.net/sensorML/1.1.0"/> 
<sch:ns prefix="swe" uri="http://www.opengis.net/swe/1.1.0"/> 
<sch:ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/> 
<sch:pattern id="IO_CHECK" name="Check I/O Characteristics"> 
<sch:rule context="&process;"> 
  <!-- CheckSum input --> 
  <sch:assert test="&input;[@name='data'] or &input;[@name='data'] and &input;[@name='checkSum']">Input named 'data' must be present</sch:assert> 
  <!-- CheckSum output --> 
  <sch:assert test="&output;[@name='data']"/Output Text named 'checkSum' must be present</sch:assert> 
  <!-- CheckSum parameter --> 
  <sch:assert test="&param;[@name='type']">Parameter named 'type' must be present</sch:assert> 
</sch:rule> 
</sch:pattern> 
<sch:pattern name="CRC_Rocksoft"> 
<sch:rule context="&param;[@name='type']/swe:DataRecord/swe:field[@name='crc']"> </sch:rule> 
<sch:assert test="swe:DataRecord/swe:field[@name='name']/swe:Text">Parameter Text named 'name' must be present in crc Block </sch:assert> 
<sch:assert test="swe:DataRecord/swe:field[@name='width']/swe:Count">Parameter Count named 'width' must be present in crc Block </sch:assert> 
<sch:assert test="swe:DataRecord/swe:field[@name='poly']/swe:Text">Parameter Text named 'poly' must be present in crc Block </sch:assert> 
<sch:assert test="swe:DataRecord/swe:field[@name='init']/swe:Text">Parameter Text named 'init' must be present in crc Block </sch:assert> 
<sch:assert test="swe:DataRecord/swe:field[@name='refin']/swe:Boolean">Parameter Boolean named 'refin' must be present in crc Block </sch:assert> 
<sch:assert test="swe:DataRecord/swe:field[@name='refout']/swe:Boolean">Parameter Boolean named 'refout' must be present in crc Block </sch:assert> 
<sch:assert test="swe:DataRecord/swe:field[@name='xorout']/swe:Text">Parameter Text named 'xorout' must be present in crc Block </sch:assert> 
<sch:assert test="swe:DataRecord/swe:field[@name='check']/swe:Text">Parameter Text named 'check' must be present in crc Block </sch:assert> 
```

B.3 Schematron rules of Interpolation Process

<!DOCTYPE schema [
<!ENTITY input "sml:inputs/sml:InputList/sml:input">
<!ENTITY output "sml:outputs/sml:OutputList/sml:output">
<!ENTITY param "sml:parameters/sml:ParameterList/sml:parameter">

<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">

<sch:pattern id="IO_CHECK" name="Check I/O Characteristics">
  <sch:rule context="&process;">
    <!-- Interpolation input -->
    <sch:assert test="&input;[@name='value']/swe:Quantity">Input Quantity named 'value' must be present</sch:assert>
    <!-- Interpolation output -->
    <sch:assert test="&output;[@name='result']/swe:Quantity">Output Quantity named 'result' must be present</sch:assert>
    <!-- Interpolation parameter -->
    <sch:assert test="&param;[@name='settings']/swe:DataArray">Parameter named 'settings' must be present</sch:assert>
    <sch:assert test="&param;[@name='settings']/swe:DataArray/swe:elementType[@name='values']/swe:DataRecord">The elementType named 'values' of parameter 'settings' must be present and contain a DataRecord element</sch:assert>
    <sch:assert test="&param;[@name='settings']/swe:DataArray/swe:elementType[@name='values']/swe:DataRecord/swe:field[@name='x']/swe:Quantity">Parameter Quantity named 'settings/x' must be present</sch:assert>
    <sch:assert test="&param;[@name='settings']/swe:DataArray/swe:elementType[@name='values']/swe:DataRecord/swe:field[@name='y']/swe:Quantity">Parameter Quantity named 'settings/y' must be present</sch:assert>
  </sch:rule>
</sch:pattern>
</sch:schema>

B.4 Schematron rules of Date Conversion Process

<!DOCTYPE schema [
<!ENTITY input "sml:inputs/sml:InputList/sml:input">
<!ENTITY output "sml:outputs/sml:OutputList/sml:output">

<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">

<sch:pattern id="IO_CHECK" name="Check I/O Characteristics">
  <sch:rule context="&process;">
    <!-- Interpolation input -->
    <sch:assert test="&input;[@name='value']/swe:Quantity">Input Quantity named 'value' must be present</sch:assert>
    <!-- Interpolation output -->
    <sch:assert test="&output;[@name='result']/swe:Quantity">Output Quantity named 'result' must be present</sch:assert>
    <!-- Interpolation parameter -->
    <sch:assert test="&param;[@name='settings']/swe:DataArray">Parameter named 'settings' must be present</sch:assert>
    <sch:assert test="&param;[@name='settings']/swe:DataArray/swe:elementType[@name='values']/swe:DataRecord">The elementType named 'values' of parameter 'settings' must be present and contain a DataRecord element</sch:assert>
    <sch:assert test="&param;[@name='settings']/swe:DataArray/swe:elementType[@name='values']/swe:DataRecord/swe:field[@name='x']/swe:Quantity">Parameter Quantity named 'settings/x' must be present</sch:assert>
    <sch:assert test="&param;[@name='settings']/swe:DataArray/swe:elementType[@name='values']/swe:DataRecord/swe:field[@name='y']/swe:Quantity">Parameter Quantity named 'settings/y' must be present</sch:assert>
  </sch:rule>
</sch:pattern>
</sch:schema>
B.5 Schematron rules for Definition of Addressing

<x:schema
  xmlns:x="http://www.ascc.net/xml/schematron"
  xmlns:xlink="http://www.w3.org/1999/xlink"
>
  <!-- CheckSum input -->
  <x:assert test="[@xlink:role='urn:ogc:def:connection:OGC:serial'] or
    [@xlink:role='urn:ogc:def:connection:OGC:tcpIP'] or
    [@xlink:role='urn:ogc:def:connection:OGC:http'] or
    [@xlink:role='urn:ogc:def:connection:OGC:udpIP'] or
    xlink:role must be present</x:assert>
</x:rule>
</x:schema>
B.6 Schematron rules for Definition of Observation Metadata

<!DOCTYPE schema [ 
<!ENTITY output "//sml:outputs/sml:OutputList/sml:output"> 
<!ENTITY offering "swe:DataRecord/gml:metaDataProperty/offering"> 
<!ENTITY foi "swe:DataRecord/gml:metaDataProperty/featureOfInterest/@xlink:href"> 
<!ENTITY observedProperty "swe:DataRecord/gml:metaDataProperty/observedProperty/@xlink:href"> 
]> 
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron"> 
<sch:title>CheckSum Process</sch:title> 
<!-- Namespaces definitions --> 
<sch:ns prefix="sml" uri="http://www.opengis.net/sensorML/1.1.0"/> 
<sch:ns prefix="swe" uri="http://www.opengis.net/swe/1.1.0"/> 
<sch:ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/> 
<sch:ns prefix="gml" uri="http://www.opengis.net/gml"/> 

<sch:pattern id="METADATA_CHECK" name="Check Observation Metadata"> 
  <sch:rule context="&output;"/> 
  <!-- CheckSum input --> 
  <sch:assert test="&offering;">offering must be present</sch:assert> 
  <!-- CheckSum output --> 
  <sch:assert test="&foi;">feature of interest must be present</sch:assert> 
  <!-- CheckSum parameter --> 
  <sch:assert test="&observedProperty;">observed property must be present</sch:assert> 
</sch:pattern> 
</sch:schema>
Annex C
(informative)

Example XML document

This annex provides the complete example of the SensorML document describing an MWS3 sensor system including its SID.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sml:System
 xmlns:sml="http://www.opengis.net/sensorML/1.1.0"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:swe="http://www.opengis.net/swe/1.1.0"
 xmlns:sid="http://www.orangenkiste.de/SID/0.5.0"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/sensorML/1.1.0 ../xsd/SID(sid.xsd"
 >
  <sml:identification>
    <sml:IdentifierList>
      <sml:identifier name="uniqueID">
        <sml:Term definition="urn:ogc:def:identifier:OGC:uniqueID"
        </sml:Term>
      </sml:identifier>
      <sml:identifier name="longName">
        <sml:Term definition="urn:ogc:def:identifier:OGC:1.0:longName"
          ><sml:value>Messwertsensor 3</sml:value>
        </sml:Term>
      </sml:identifier>
    </sml:IdentifierList>
  </sml:identification>
  <sml:capabilities>
    <swe:DataRecord definition="urn:ogc:def:property:OGC::status">
      <!-- station is collecting data -->
      <swe:field name="status">
        <swe:Boolean><swe:value>true</swe:value></swe:Boolean>
      </swe:field>
      <swe:field name="mobile">
        <swe:Boolean><swe:value>false</swe:value></swe:Boolean>
      </swe:field>
      <swe:field name="measuringInterval">
        <swe:Quantity definition="urn:ogc:def:property:OGC:1.0:measuringInterval"
          >The measuring interval of the MWS3</swe:Quantity>
        <swe:uom code="ms"/>
        <swe:value>1000</swe:value>
      </swe:field>
    </swe:DataRecord>
  </sml:capabilities>
  <sml:position name="Position">
    <swe:Position definition="urn:ogc:def:property:OGC:1.0:stationPosition"
      referenceFrame="urn:ogc:def:crs:EPSG:4326">
      <swe:location>
        <swe:Vector
          ><swe:coordinate name="latitude">
            <swe:Quantity axisID="y"/>
            <swe:uom code="deg"/>
          </swe:coordinate>
          
```
\[
\begin{aligned}
&\text{swe:constraint} \\
&\text{swe:AllowedValues} \\
&\text{swe:interval} = -180 \text{ to } 180 \\
&\text{swe:constraint} \\
&\text{swe:value} = 7.2 \\
&\text{swe:Quantity} \\
&\text{name} = \text{Geodetic longitude} \\
&\text{swe:coordinate} \\
&\text{swe:AllowedValues} \\
&\text{swe:interval} = -180 \text{ to } 180 \\
&\text{swe:constraint} \\
&\text{swe:value} = 52 \\
&\text{swe:Quantity} \\
&\text{swe:Vector} \\
&\text{swe:location} \\
&\text{sml:Position} \\
&\text{sml:interfaces} \\
&\text{sml:InterfaceList} \\
&\text{sml:interface} \\
&\text{command} \\
&\text{swe:ParameterDefinition} \\
&\text{sme:CommandDefinition} \\
&\text{sml:applicationLayer} \\
&\text{sid:CommandDefinition} \\
&\text{sid:commands} \\
&\text{command name} = \text{setSamplingRateCommand} \\
&\text{sid:CommandParameters} \\
&\text{name} = \text{setSamplingRateCommandParameters} \\
&\text{swe:DataRecord} \\
&\text{swe:field name} = \text{commandName} \\
&\text{swe:Text} \\
&\text{swe:value} = \text{SR} \\
&\text{swe:Text} \\
&\text{swe:field name} = \text{space} \\
&\text{swe:Text} \\
&\text{swe:value} \\
&\text{swe:Text} \\
&\text{swe:field name} = \text{measuringInterval} \\
&\text{swe:Count} \\
&\text{swe:DataRecord} \\
&\text{response valid} = \text{true} \\
&\text{ackResponse} \\
&\text{swe:DataRecord} \\
&\text{ack} \\
&\text{swe:DataRecord} \\
&\text{nackResponse} \\
&\text{swe:DataRecord} \\
&\text{nack} \\
\end{aligned}
\]
<!-- incoming data stream: -->
<sid:DataInputStream>
<dataInputComponents>
<ComponentList>
  <component name="samplingRateCommand">
    <swe:DataRecord>
      <swe:field name="command"/>
    </swe:DataRecord>
  </component>
</ComponentList>
</dataInputComponents>
</sid:DataInputStream>

<!-- outgoing data stream: -->
<sid:DataOutputStream>
<dataOutputComponents>
<ComponentList>
  <component name="ackResponse">
    <swe:DataRecord>
      <swe:field name="ack"/>
    </swe:DataRecord>
  </component>
  <component name="nackResponse">
    <swe:DataRecord>
      <swe:field name="nack"/>
    </swe:DataRecord>
  </component>
  <component name="odl_basis_01">
    <swe:DataRecord>
      <!-- ~~~~~~~~~~~~~~~~~~~ sampling time ~~~~~~~~~~~~~~~~~~~ -->
      <swe:field name="samplingTime">
        <swe:Time>
          <gml:description>time when a data set was measured</gml:description>
          <swe:uom code="ISO8601"/>
        </swe:Time>
      </swe:field>
    </swe:DataRecord>
    <!-- ~~~~~~~~~~~~~~~~~~~ temperature ~~~~~~~~~~~~~~~~~~~ -->
    <swe:field name="temp"/>
  </component>
</ComponentList>
</dataOutputComponents>
</sid:DataOutputStream>

<-- incoming data stream: -->
<sid:ResponseList>
</command>
</sid:CommandList>
</sid:Commands>
</sid:CommandDefinition>
</sml:applicationLayer>
</sml:presentationLayer>

<!-- incoming data stream: -->
<sid:DataInputStream>
<dataInputComponents>
<ComponentList>
  <component name="samplingRateCommand">
    <swe:DataRecord>
      <swe:field name="command"/>
    </swe:DataRecord>
  </component>
</ComponentList>
</dataInputComponents>
</sid:DataInputStream>

<!-- outgoing data stream: -->
<sid:DataOutputStream>
<dataOutputComponents>
<ComponentList>
  <component name="ackResponse">
    <swe:DataRecord>
      <swe:field name="ack"/>
    </swe:DataRecord>
  </component>
  <component name="nackResponse">
    <swe:DataRecord>
      <swe:field name="nack"/>
    </swe:DataRecord>
  </component>
  <component name="odl_basis_01">
    <swe:DataRecord>
      <!-- ~~~~~~~~~~~~~~~~~~~ sampling time ~~~~~~~~~~~~~~~~~~~ -->
      <swe:field name="samplingTime">
        <swe:Time>
          <gml:description>time when a data set was measured</gml:description>
          <swe:uom code="ISO8601"/>
        </swe:Time>
      </swe:field>
    </swe:DataRecord>
    <!-- ~~~~~~~~~~~~~~~~~~~ temperature ~~~~~~~~~~~~~~~~~~~ -->
    <swe:field name="temp"/>
  </component>
</ComponentList>
</dataOutputComponents>
</sid:DataOutputStream>
<swe:Quantity gml:id="temperatureData">
  <gml:description>temperature</gml:description>
  <swe:field>
    <swe:uom code="CEL"/>
  </swe:field>
</swe:Quantity>

------------------------

<swe:field name="humidity">
  <swe:Quantity gml:id="humidityData">
    <gml:description>humidity</gml:description>
    <swe:uom code="g/m3"/>
  </swe:Quantity>
</swe:field>

<swe:DataRecord>
  <component>
    ...<ComponentList>
    ...<dataOutputComponents>
    ...<presentationLayer>
    ...<transportLayer>
    ...<Decoder>
    ...<decoderComponent>
    ...<ProcessChain>
    ...<input name="resistance_hd">
    ...<swe:Quantity/>
    ...</input>
    ...<output name="radiation_hd">
    ...<swe:Quantity/>
    ...</output>
    ...<ParameterList>
    ...<parameter name="interpolationParameter">
    ...<DataRecord>
    ...<field name="resistance">
    ...<swe:Quantity/>
    ...</field>
    ...<field name="radiation">
    ...<swe:Quantity/>
    ...</field>
    ...</DataRecord>
    ...</parameter>
    ...</ParameterList>
    ...<components>
    ...<ComponentList>
    ...<component name="interpolationProcess_hd">
    ...<ProcessModel gml:id="interpolationProcess">
    ...<inputs>
    ...<input name="value">
    ...<swe:Quantity/>
    ...</input>
    ...</inputs>
    ...<outputs>
    ...</outputs>
    ...</ComponentList>
    ...</component>
    ...</ComponentList>
    ...</components>
    ...</dataOutputComponents>
    ...</ComponentList>
    ...</presentationLayer>
    ...</transportLayer>
    ...</Decoder>
    ...</decoderComponent>
    ...</ProcessChain>
  </component>
</swe:DataRecord>
<sml:output name="result">
  <swe:Quantity/>
</sml:output>
</sml:OutputList>
</sml:outputs>
<sml:parameters>
  <sml:ParameterList>
    <sml:parameter name="settings">
      <swe:DataArray>
        <swe:elementCount/>
        <swe:elementType name="values">
          <swe:DataRecord>
            <swe:field name="x">
              <swe:Quantity/>
            </swe:field>
            <swe:field name="y">
              <swe:Quantity/>
            </swe:field>
          </swe:DataRecord>
        </swe:elementType>
      </swe:DataArray>
    </sml:parameter>
  </sml:ParameterList>
  <sml:parameters>
  </sml:method>
</sml:parameters>
<sml:connections>
  <sml:ConnectionList>
    <sml:connection>
      <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="this/inputs/resistance_hd"/>
        <sml:destination/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="interpolationProcess_hd/inputs/value"/>
        <sml:destination/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="interpolationProcess_hd/outputs/result"/>
        <sml:destination/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="this/outputs/radiation_hd"/>
        <sml:destination/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="this/parameters/interpolationParameter/resistance"/>
        <sml:destination/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="interpolationProcess_hd/parameter/settings/values/x"/>
        <sml:destination/>
      </sml:Link>
    </sml:connection>
  </sml:ConnectionList>
</sml:connections>
</sml:ComponentList>
</sml:components>
</sml:ComponentList>
</sml:ProcessModel>
<swe:Text definition="urn:ogc:def:TimeString"/>
</sml:input>
</sml:InputList>
</sml:inputs>

<xsml:outputs>

<xsml:output name="date">
<swe:Text definition="urn:ogc:def:TimeString"/>
</xsml:output>
</xsml:OutputList>
</xsml:outputs>

<xsml:parameters>
<xsml:ParameterList>

<xsml:parameter name="dateSettings">
<swe:DataRecord>
<swe:field name="inputFormat"/>
<swe:field name="outputFormat"/>
</swe:DataRecord>
</xsml:parameter>
</xsml:ParameterList>
</xsml:parameters>

<xsml:method xlink:href="dateConversion.xml#123" xlink:role="urn:ogc:def:process:OGC:1.0:dateConversion"/>
</xsml:ProcessModel>
</xsml:component>

<xsml:ComponentList>
<xsml:connections>

<xsml:connection>
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="this/inputs/date01"/>
<sml:destination ref="dateConversion_01/inputs/date"/>
</sml:Link>
</xsml:connection>

<xsml:connection>
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="dateConversion_01/outputs/date"/>
<sml:destination ref="this/outputs/date01"/>
</sml:Link>
</xsml:connection>

<xsml:connection>
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="dateConversion_10/inputs/date"/>
<sml:destination ref="this/inputs/date10"/>
</sml:Link>
</xsml:connection>

<xsml:connection>
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="dateConversion_10/outputs/date"/>
<sml:destination ref="this/outputs/date10"/>
</sml:Link>
</xsml:connection>

<xsml:connection>
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="dateConversion_01/inputs/date"/>
<sml:destination ref="dateConversion_01/outputs/date"/>
</sml:Link>
</xsml:connection>

<xsml:connection>
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="dateConversion_10/inputs/date"/>
<sml:destination ref="dateConversion_10/outputs/date"/>
</sml:Link>
</xsml:connection>
</xsml:connections>
</xsml:ComponentList>
</xsml:components>

<sml:component name="dateConversion_01" xlink:href="#dateConversion"/>
<sml:component name="dateConversion_10" xlink:href="#dateConversion"/>
<swe:field xlink:role="urn:ogc:def:encoding:assertedValue" name="ack">
  <swe:Text>
    <swe:value>0x06</swe:value>
  </swe:Text>
</swe:field>
<byteOrder="bigEndian" />
<byteLength="1" />
</swe:member>
</swe:BinaryBlock>
</swe:encoding>
</swe:DataBlockDefinition>
</component>
</component name="nackResponse">
</swe:components name="data">
<swe:DataRecord>
  <swe:field xlink:role="urn:ogc:def:encoding:assertedValue" name="nack">
    <swe:Text>
      <swe:value>0x15</swe:value>
    </swe:Text>
  </swe:field>
</swe:DataRecord>
<byteOrder="bigEndian" />
<byteLength="1" />
</swe:member>
</swe:BinaryBlock>
</swe:encoding>
</swe:DataBlockDefinition>
</component name="M01">
</swe:components name="data">
<swe:DataRecord>
  <swe:field name="id">
    <swe:Text description="data block identifier">
      <swe:value>M01</swe:value>
    </swe:Text>
  </swe:field>
  <swe:field name="timeTag">
    <swe:Count description="time tag of the measurement in ms">
      <swe:value>0</swe:value>
    </swe:Count>
  </swe:field>
</swe:DataRecord>
<!-- ~~~~~~~~~~~~~~~~ high dose resistance ~~~~~~~~~~~~~~~-->
<swf:field name="imp_hd_resistance">
  <swf:Quantity>
    <gml:description>resistance of high dose radiation detector</gml:description>
  </swf:Quantity>
</swf:field>

<!-- ~~~~~~~~~~~~~~~~ temperature ~~~~~~~~~~~~~~~-->
<swf:field name="temp">
  <swf:Quantity>
    <gml:description>temperature</gml:description>
  </swf:Quantity>
</swf:field>

<!-- ~~~~~~~~~~~~~~~~ humidity ~~~~~~~~~~~~~~~-->
<swf:field name="humidity">
  <swf:Quantity>
    <gml:description>humidity</gml:description>
  </swf:Quantity>
</swf:field>

</swe:DataRecord>
</swe:components>
</swe:encoding>
</swe:DataBlockDefinition>
</component>
</dataOutputComponents>
</sml:physicalLayer>
</sml:connections>
</sml:ConnectionList>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~ Connections for ack DataSet ~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~ Connections for incoming setSamplingRateCommand: ~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- ~-------------------------------- Linking of incoming setSamplingRateCommand: ~----------------------------------->
<sml:connection>
  <sml:Link type="urn:ogc:def:link:OGC:sid">
    <sml:source ref="applicationLayer/commands/setSamplingRateCommand"/>
    <sml:destination ref="presentationLayer/dataInputComponents/samplingrateCommand/"
                    </sml:Link>
    <sml:connection>
      <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="presentationLayer/dataInputComponents/samplingrateCommand/setSamplingRateCommandParameters"/>
        <sml:destination ref="presentationLayer/dataInputComponents/samplingrateCommand/"/>
<sml:destination ref="physicalLayer/dataInputComponents/samplingrateCommand"/>
</sml:Link>
</sml:connection>

<!-- Linking of outgoing 'ack' response (response to setSamplingRateCommand): -->

</sml:connection>

<!-- Linking of outgoing 'ack' response (response to setSamplingRateCommand): -->
<
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="physicalLayer/dataOutputComponents/ackResponse/data/ack"/>
<sml:destination ref="presentationLayer/dataOutputComponents/ackResponse/ack"/>
</sml:Link>
</sml:connection>
<
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="presentationLayer/dataOutputComponents/ackResponse/ack"/>
<sml:destination ref="applicationLayer/commands/setSamplingRateCommand/ackResponse/ack"/>
</sml:Link>
</sml:connection>

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~

----------------------------------
M01 DataSet ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
----------------------------------

----------------------------------
Connections for high dose
----------------------------------

----------------------------------
Linking time tag through date conversion process (networkLayer) to presentationLayer: -->
<
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="networkLayer/decoderComponent/inputs/date01"/>
<sml:destination ref="networkLayer/decoderComponent/outputs/date01"/>
</sml:Link>
</sml:connection>
<
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="networkLayer/decoderComponents/inputs/resistance_hd"/>
<sml:destination ref="presentationLayer/dataOutputComponents/odl_basis_01/samplingTime"/>
</sml:Link>
</sml:connection>

----------------------------------

----------------------------------
Linking high dose radiation data through interpolation process (transportLayer) to presentationLayer: -->
<
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="transportLayer/decoderComponents/inputs/resistance_hd"/>
<sml:destination ref="physicalLayer/dataOutputComponents/M01/data/imp_hd_resistance"/>
</sml:Link>
</sml:connection>
<
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="physicalLayer/dataOutputComponents/ackResponse/data/ack"/>
<sml:destination ref="physicalLayer/dataOutputComponents/ackResponse/data/ack"/>
</sml:Link>
</sml:connection>
<sml:source ref="transportLayer/decoderComponents/outputs/radiation_hd"/>
<sml:destination ref="presentationLayer/dataOutputComponents/odl_basis_01/highDoseRadiation"/>
</sml:Link>
</sml:connection>

<!-- Direct linking of physicalLayer::dataOutputComponents to presentationLayer::dataOutputComponents -->
<sml:connection>
<sml:Link type="urn:ogc:def:link:OGC:sid">
<sml:source ref="physicalLayer/dataOutputComponents/M01/data/temp"/>
<sml:destination ref="presentationLayer/dataOutputComponents/odl_basis_01/temp"/>
</sml:Link>
</sml:connection>
</sml:connection>
</sml:connectionList>
</sml:connections>
</sml:InterfaceDefinition>
</sml:InterfaceList>
</sml:interfaces>

<sml:inputs/>
<sml:outputs>
<sml:OutputList>

<sml:output name="highDoseRadiation_output">
<swe:DataRecord>
<gml:metaDataProperty>
<offering>RADIATION_OFFERING</offering>
</gml:metaDataProperty>
<gml:metaDataProperty>
<featureOfInterest xlink:href="http://myWFS.org/features/Muenster_city"/>
</gml:metaDataProperty>
<observedProperty xlink:href="urn:ogc:def:property:OGC:radiation"/>
<swe:field name="dataSet" xlink:href="#highDoseRadiationData" />
</swe:DataRecord>
</sml:output>

<sml:output name="temperature_output">
<swe:DataRecord>
<gml:metaDataProperty>
<offering>TEMPERATURE_OFFERING</offering>
</gml:metaDataProperty>
<gml:metaDataProperty>
<featureOfInterest xlink:href="http://myWFS.org/features/Muenster_city"/>
</gml:metaDataProperty>
<observedProperty xlink:href="urn:ogc:def:property:OGC:temperature"/>
</swe:DataRecord>
</sml:output>

...
<swe:DataRecord name="dataSet" xlink:href="#temperatureData" />
<!-- this xlink:href includes the element with that gml:id as a sub-element here -->
</swe:DataRecord>

</sml:output>
<sml:output name="humidity_output">
<swe:DataRecord>
<gml:metaDataProperty>
<offering>HUMIDITY_OFFERING_01</offering>
</gml:metaDataProperty>
<featureOfInterest>
<xlink:href="http://myWFS.org/features/Muenster_city"/>
</featureOfInterest>
<observedProperty>
</observedProperty>
<swe:field name="dataSet" xlink:href="#humidityData" />
</swe:DataRecord>
</sml:output>
</sml:OutputList>
</sml:outputs>

<sml:components>
<sml:ComponentList>
<sml:component name="radiometer">
<sml:Component>
<sml:parameters>
<sml:ParameterList>
<sml:parameter name="steadyStateCurve">
<swe:Curve>
<swe:elementCount>
<swe:Count>
<swe:value>2</swe:value>
</swe:Count>
</swe:elementCount>
<swe:elementType>
<swe:SimpleDataRecord>
<swe:field name="resistance">
</swe:field>
</swe:SimpleDataRecord>
</swe:elementType>
<swe:encoding>
<swe:TextBlock tokenSeparator="", decimalSeparator=".", blockSeparator="; " />
</swe:encoding>
<swe:values>-3,-5; 6,7; 8.2,7.5</swe:values>
</swe:Curve>
</sml:parameter>
</sml:ParameterList>
</sml:parameters>
</sml:Component>
</sml:component>
</sml:ComponentList>
</sml:components>
<!-- Instantiation of SID internal interpolation process. Linking of externally defined process parameters to parameters of interpolation process: -->
<sml:connection>
    <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="components/barometer/parameters/steadyStateCurve/resistance" />
        <sml:destination ref="this/interfaces/mws/transportLayer/decoderComponent/parameters/interpolationParameter/resistance" />
    </sml:Link>
</sml:connection>

<!-- Linking presentationLayer::dataOutputComponents to outputs: -->
<sml:connection>
    <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="this/interfaces/mws/presentationLayer/dataOutputComponents/odl_basis_01/highDoseRadiation" />
        <sml:destination ref="this/outputs/highDoseRadiation_output/dataSet" />
    </sml:Link>
</sml:connection>

<sml:connection>
    <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="this/interfaces/mws/presentationLayer/dataOutputComponents/odl_basis_01/temp" />
        <sml:destination ref="this/outputs/temperature_output/dataSet" />
    </sml:Link>
</sml:connection>

<sml:connection>
    <sml:Link type="urn:ogc:def:link:OGC:sid">
        <sml:source ref="this/interfaces/mws/presentationLayer/dataOutputComponents/odl_basis_01/humidity" />
        <sml:destination ref="this/outputs/humidity_output/dataSet" />
    </sml:Link>
</sml:connection>

</sml:ConnectionList>
</sml:connections>
</sml:System>
Bibliography


