Open Geospatial Consortium

Publication Date: 2014-02-04

Approval Date: 2013-12-10

Submission Date: 2012-11-05

Reference number of this OGC® document: OGC 12-000

OGC name of this OGC® project document: http://www.opengis.net/doc/IS/SensorML/2.0

Version: 2.0.0

Category: OGC® Encoding Standard

Editor: Mike Botts
Co-Editor: Alexandre Robin

OGC® SensorML: Model and XML Encoding Standard

Warning

Document type: OGC® Publicly Available Standard
Document subtype: Encoding
Document stage: Approved
Document language: English

License Agreement

Contents

Table of Figures

i. Abstract

The primary focus of the Sensor Model Language (SensorML) is to provide a robust and semantically-tied means of defining processes and processing components associated with the measurement and post-measurement transformation of observations. This includes sensors and actuators as well as computational processes applied pre- and post-measurement.

The main objective is to enable interoperability, first at the syntactic level and later at the semantic level (by using ontologies and semantic mediation), so that sensors and processes can be better understood by machines, utilized automatically in complex workflows, and easily shared between intelligent sensor web nodes.

This standard is one of several implementation standards produced under OGC's Sensor Web Enablement (SWE) activity. This standard is a revision of content that was previously integrated in the SensorML version 1.0 standard (OGC 07-000).

ii. Keywords

ogcdoc, sensor model language, sensorml, swe, sensors, actuators, detectors, transducers, processes, processing, observations, measurement, data quality, data lineage, data provenance

iii. Submitting Organizations

The following organizations have contributed and submitted this Encoding Standard to the Open Geospatial Consortium:

- Botts Innovative Research, Inc.
- Sensia Software LLC
- Spot Image, S.A.
- Seicorp, Inc.
- US National Geospatial Intelligence Agency (NGA)
- TASC

Submission Contact Points

All questions regarding this submission should be directed to the editor or the submitters:

Contact	Company	Email
Michael E. Botts	Botts Innovative Research, Inc	mike.botts<at>botts-inc.com
Alexandre Robin	Sensia Software LLC	alex.robin<at>sensiasoftware.com
Jim Greenwood	Seicorp, Inc.	jgreenwood<at>Seicorp.com
David Wesloh	US National Geospatial Intelligence Agency	David.G.Wesloh<at>nga.mil

iv. Future Work

Future work will target the description of specialized processes, sensors, and actuators by restricting (via profiles) the generic models and schema defined in this standard. Such profiles will allow interoperability of models and schema both within specific processing and measurement communities as well as between disparate.

v. Changes to the OGC® Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate this OGC® Standard.

Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights. However, to date, no such rights have been claimed or identified. 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 specification set forth in this document, and to provide supporting documentation.

Introduction

This standard originated from work originally undertaken through the Open Geospatial Consortium's Sensor Web Enablement (SWE) activity. SWE is concerned with establishing interfaces and encodings that will enable a "Sensor Web" through which applications and services will be able to access sensors of all types, and observations generated by them, over the Web. SWE has defined, prototyped and tested several components needed for a Sensor Web, namely:

- Sensor Model Language (SensorML)
- Observations & Measurements (O&M)
- Sensor Observation Service (SOS)
- Sensor Planning Service (SPS)
- SWE Common Data and Services

This standard specifies models and an XML implementation for the SensorML.
This document deprecates and replaces the first version of OGC® Sensor Model Language (SensorML) Specification version 1.0.0 (OGC 07-000) and the SensorML Corrigendum version 1.0.1 (OGC 07-122r2).
The main changes from SensorML version 1.0.1 are:

- The separation of SWE Common Data into a separate specification (OGC 07-094r1)
- Improved derivation and association of SensorML from GML 3.2 and ISO 19115
- More explicit definition of the standard and its requirements
- Separation of SensorML into several conformance classes to allow software to support only the part of SensorML that is relevant to the application (e.g. non-physical processes only)
- Improved support for inheritance, configuration, and modes (e.g. for describing a particular model of a sensor and then an instance of that sensor with particular configuration)
- Improved explicit support for data streaming (associated with inputs, outputs, and parameters)
- Addition of Feature of Interest for support of discovery
- Addition of extension points for domain or community-specific schema
- Improved support for defining position of both static and dynamic components and systems
- Inclusion of DataInterface and DataStream as a possible input, output, or parameter value

Additionally, much additional and improved functionality of SensorML has been gained through additions and improvements of the SWE Common Data Model specification.

Thus, the following additions to SWE Common Data Model are reflected as improvements in SensorML v2.0:

- A DataChoice component providing support for variable (multiplexed) data types
- The DataStream object improving support for real-time data streams
- The XMLBlock encoding providing support for simple XML encoded data
- Support for definition of NIL values and associated reasons
- The CategoryRange class to define ranges of ordered categorical quantities
- Extension points for domain or community-specific schema
- Ability to provide security tagging of individual data components through the use of extension points

Also, some elements of the SWE Common Data Model specification have been removed and replaced by their soft-typed equivalents defined using RelaxNG and/or Schematron. These include:

- Position, SquareMatrix
- SimpleDataRecord, ObservableProperty
- ConditionalData, ConditionalValue
- Curve, NormalizedCurve

The derivation from GML has also been improved by making all elements substitutable for GML AbstractValue (and thus transitively for GML AbstractObject) and AbstractFeature so that they can be used directly by GML application schemas. The GML encoding rules as defined in ISO 19136 have also been used to generate XML schemas from the UML models with only minor modifications.
This release is not fully backwards compatible with version 1.0.1 even though changes were kept to a minimum.

Sensor Model Language (SensorML): An Implementation Standard

1 Scope

This standard defines models and XML Schema encoding for SensorML. The primary focus of SensorML is to provide a framework for defining processes and processing components associated with the measurement and post-measurement transformation of observations. Thus SensorML has more of a focus on the process of measurement and observation, rather than on sensor hardware, yet still provides a robust means of defining the physical characteristics and functional capabilities of physical processes such as sensors and actuators.

The aims of SensorML are to:

- Provide descriptions of sensors and sensor systems for inventory management
- Provide sensor and process information in support of asset and observation discovery
- Support the processing and analysis of the sensor observations
- Support the geolocation of observed values (measured data)
- Provide performance and quality of measurement characteristics (e.g., accuracy, threshold, etc.)
- Provide general descriptions of components (e.g. a particular model or type of a sensor) as well as the specific configuration of that component when its deployed
- Provide a machine interpretable description of the interfaces and data streams flowing in and out of a component
- Provide an explicit description of the process by which an observation was obtained (i.e., it's lineage)
- Provide an executable aggregate process for deriving new data products on demand (i.e., derivable products)
- Archive fundamental properties and assumptions regarding sensor systems and computational processes

SensorML provides a common framework for any process, but is particularly well-suited for the description of sensor and systems and the processes surrounding sensor observations. Within SensorML, sensor and transducer components (detectors, transmitters, actuators, and filters) are all modeled as physical processes that can be connected and participate equally within a process network or system, and which utilize the same model framework as any other process.

Processes are entities that take one or more inputs and through the application of well-defined methods and configurable parameters, and produce one or more outputs. The process model defined in SensorML can be used to describe a wide variety of processes,

including not only sensors, but also actuators, spatial transforms, and data processes, to name a few. SensorML also supports explicit linking between processes and thus supports the concept of process chains, networks, or workflows, which are themselves defined as processes using a composite pattern.

SensorML provides a framework within which the geometric, dynamic, and observational characteristics of sensors and sensor systems can be defined. There are a great variety of sensor types, from simple thermometers to complex electron microscopes and earth observing satellites. These can all be supported through the definition of simple and aggregate processes.

The models and schema within the core SensorML specification provide a "skeletal" framework for describing processes, aggregate processes, and sensor systems. Interoperability within and between various sensor communities, is greatly improved through the definition of shared community-specific semantics (within online dictionaries or ontologies) that can be utilized within the framework. In addition, the profiling of small, general-use, atomic processes that can serve as components within aggregate processes and systems is envisioned.

2 Conformance

2.1 Overview

This standard has been written to be compliant with the OGC Specification Model – A Standard for Modular Specification (OGC 08-131r3). Extensions of this standard should themselves be conformant to the OGC Specification Model.

This standard defines conceptual models and an XML implementation of these models for describing non-physical and physical processes surrounding the act of measurement and subsequent processing of observations. The conceptual models are described using UML while the implementation is described using the XML Schema language and Schematron. It is envisioned that OWL (Web Ontology Language) and RDF (Resource Description Framework) versions could also be generated from the models if deemed useful and desired.

2.2 Specification identifier

All requirements-classes and conformance-classes described in this document are owned by the specification identified as **http://www.opengis.net/spec/sensorml/2.0**.

2.3 Conformance classes

The conformance rules are based on XML validation using the XML Schema representation of SensorML, together with processing of constraints expressed using Schematron assertions and reports.

Conformance with this specification shall be checked using all the relevant tests specified in Annex A. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in ISO 19105: Geographic information — Conformance and Testing. In order to conform to this OGC™ encoding standard, a standardization target shall implement the core conformance class, and choose to implement any one or more of the other conformance classes.

The conformance rules for the XML implementation are based on XML validation using XML Schema representation of SensorML, together with processing constraints expressed using Schematron assertions and reports.

Table 1 — Conformance classes related SensorML instances

Requirements class	Description	Clause
Core Concepts		
http://www.opengis.net/spec/sensorml/2.0/req/core	Core Concepts	A.1
Conceptual Models:		
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process	Abstract Process	A.2
http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process	(Non-Physical) Simple Process	A.3
http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process	(Non-Physical) Aggregate Process	A.4
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component	Physical Component	A.5
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system	Physical System	A.6
http://www.opengis.net/spec/sensorml/2.0/req/model/advanced-processes	Processes with Advanced Data Types	A.7
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process	Configurable Process	A.8
XML Schema:		
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process	Core Process	A.9
http://www.opengis.net/spec/sensorml/2.0/req/xml/simple-process	(Non-Physical) Simple Process	A.10
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process	(Non-Physical) Aggregate Process	A.11
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component	Physical Component	A.12
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-system	Physical System	A.13
http://www.opengis.net/spec/sensorml/2.0/req/xml/configurable-process	Configurable Process	A.14

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of document OGC 08-094. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

- OGC 08-131r3 – The Specification Model – A Standard for Modular Specification
- OGC 08-094r1 – SWE Common Data Model Encoding Standard, version 2.0
- ISO/IEC 11404:2007 – General-Purpose Datatypes
- ISO 8601:2004 – Representation of Dates and Times
- ISO 19103:2005 – Conceptual Schema Language
- ISO 19108:2002 – Temporal Schema
- ISO 19111:2007 – Spatial Referencing by Coordinates
- ISO 19115:2006 – All Metadata
- ISO 19136 - GML
- Unified Code for Units of Measure (UCUM) – Version 1.8, July 2009
- Unicode Technical Std #18 – Unicode Regular Expressions, Version 13, Aug. 2009
- The Unicode Standard, Version 5.2, October 2009
- W3C Extensible Markup Language (XML) – Version 1.0 (4th Edition), Aug. 2006
- W3C XML Schema – Version 1.0 (Second Edition), October 2004
- IEEE 754:2008 – Standard for Binary Floating-Point Arithmetic
- IETF RFC 2045 – Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, November 1996
- IETF RFC 5234 – Augmented BNF for Syntax Specifications: ABNF

4 Terms and Definitions

For the purpose of this document, the following terms and definitions apply:

4.1. Actuator

A type of transducer that converts a signal to some real-world action or phenomenon.

4.2. Aggregate Process

Composite process consisting of interconnected sub-processes, which can in turn be **Simple Processes** or themselves **Aggregate Processes**. An aggregate process can include possible data sources. A decription of an aggregate process should explicitly define connections that link input and output signals of sub-processes together. Since it is a process itself, an aggregate process also has its own inputs, outputs and parameters.

4.3. Coordinate Reference System (CRS)

A spatial or temporal framework within which a position and/or time can be defined. According to ISO 19111, a **coordinate system** that is related to the real world by a **datum**.

4.4. Coordinate System (CS)

According to ISO19111, a set of (mathematical) rules for specifying how coordinates are assigned to points. In this document, a Coordinate System is extended to be defined as a set of axes with which location and orientation can be defined.

4.5. Data Component

Element of sensor data definition corresponding to an atomic or aggregate data type

Note: A data component is a part of the overall dataset definition. The dataset structure can then be seen as a hierarchical tree of data components.

4.6. Datum

Undefined in ISO 19111. Defined here as a means of relating a **coordinate system** to the real world by specifying the physical location of the coordinate system and the orientation of the axes relative to the physical object. For a geodetic datum, the definition also includes a reference ellipsoid that approximates the physical or gravitational surface of the planetary body.

4.7. Detector

Atomic part of a composite **Measurement System** defining sampling and response characteristic of a simple detection device. A detector has only one input and one output, both being scalar quantities. More complex **Sensors**, such as a frame camera, which are composed of multiple detectors, can be described as a detector group or array using a **System** or **Sensor** model.

4.8. Determinand

A Parameter or a characteristic of a phenomenon subject to **observation.** Synonym for **observable.** *[O&M]*

4.9. Feature

Abstraction of real-world phenomena [ISO 19101:2002, definition 4.11]

Note: A feature may occur as a type or an instance. Feature type or feature instance should be used when only one is meant.

4.10. Location
A point or extent in space relative to a **coordinate system**. For point-based systems, this is typical expressed as a set of n-dimensional coordinates within the **coordinate system**. For bodies, this is typically expressed by relating the translation of the origin of an object's local **coordinate system** with respect to the origin of an external reference **coordinate system**.

4.11. Location Model
A model that allows one to locate objects in one local **reference frame** relative to another **reference frame**.

4.12. Measurand
Physical parameter or a characteristic of a phenomenon subject to a **measurement,** whose value is described using a Measure (ISO 19103). Subset of **determinand** or **observable**. *[O&M]*

4.13. Measure (noun)
Value described using a numeric amount with a scale or using a scalar reference system *[ISO/TS 19103]*. When used as a noun, measure is a synonym for physical quantity

4.14. Measurement (noun)
An **observation** whose result is a **measure** *[O&M]*

4.15. Measurement (verb)
An instance of a procedure to estimate the value of a natural phenomenon, typically involving an instrument or sensor. This is implemented as a dynamic feature type, which has a property containing the result of the measurement. The measurement feature also has a location, time, and reference to the method used to determine the value. A measurement feature effectively binds a value to a location and to a method or instrument.

4.16. Muliplexed Data Stream
A data stream that consists of disparate but well defined data packets within the same stream.

4.17. Observable, Observable Property (noun)
A parameter or a characteristic of a **phenomenon** subject to **observation.** Synonym for **determinand.** *[O&M]*

A physical property of a phenomenon that can be observed and measured (e.g. temperature, gravitational force, position, chemical concentration, orientation, number-of-individuals, physical switch status, etc.), or a characteristic of one or more feature types, the value for which will be estimated by application of some procedure in an observation. It is thus a physical stimulus that can be sensed by a detector or created by an actuator.

4.18. Observation
Act of observing a property or phenomenon [ISO/DIS 19156, definition 4.10]
Note: The goal of an observation may be to measure, estimate or otherwise determine the value of a property.

4.19. Observation Procedure
Method, algorithm or instrument, or system of these which may be used in making an observation [ISO/DIS 19156, definition 4.11]
Note: In the context of the sensor web, an observation procedure is often composed of one or more sensors that transform a real world phenomenon into digital information, plus additional processing steps.

4.20. Observed Value
A **value** describing a natural phenomenon, which may use one of a variety of scales including nominal, ordinal, ratio and interval. The term is used regardless of whether the value is due to an instrumental observation, a subjective assignment or some other method of estimation or assignment. [O&M]

4.21. Orientation
The rotational relationship of an object relative to an external coordinate system. Typically expressed by relating the rotation of an object's local coordinate axes relative to those axes of an external reference coordinate system.

4.22. Phenomenon
A physical state that can be observed and its properties measured.

4.23. Physical System
An aggregate model of a group or array of process components, which can include detectors, actuators, or sub-systems. A **Physical System** relates an **Aggregate Process** to the real world and therefore provides additional definitions regarding relative positions of its components and communication interfaces.

4.24. Position
The location and orientation of an object relative to an external coordinate system. For body-based systems (in lieu of point-based systems) is typically expressed by relating the object's local coordinate system to an external reference coordinate system. This definition is in contrast to some definitions (e.g. ISO 19107) which equate position to location.

4.25. Process
An operation that takes one or more inputs, and based on a set of parameters, and a methodology generates one or more outputs.

4.26. Process Method
Definition of the algorithm, behaviour, and interface of a **Process**.

4.27. Property
Facet or attribute of an object referenced by a name
[ISO/DIS 19143:2010]
Example: Abby's car has the color red, where "color" is a property of the car instance, and "red" is the value of that property.

4.28. Reference Frame
A coordinate system by which the position (location and orientation) of an object can be referenced.

4.29. Result
An estimate of the value of some property generated by a known procedure [O&M]

4.30. Sample
A representative subset of the physical entity on which an observation is made.

4.31. Sensor
An entity capable of observing a phenomenon and returning an observed value. Type of observation procedure that provides the estimated value of an observed property at its output.
Note: A sensor uses a combination of physical, chemical or biological means in order to estimate the underlying observed property. At the end of the measuring chain electronic devices often produce signals to be processed.

4.32. Sensor Model
In line with traditional definitions of the remote sensing community, a sensor model is a type of Location Model that allows one to georegister or co-register observations from a sensor (particularly remote sensors).

4.33. Sensor Data
List of digital values produced by a sensor that represents estimated values of one or more observed properties of one or more features.
Note: Sensor data is usually available in the form of data streams or computer files.

4.34. Sensor-Related Data
List of digital values produced by a sensor that contains ancillary information that is not directly related to the value of observed properties
Example: sensor status, quality of measure, quality of service, battery life, etc. Such data can be sent in the same data stream with measured values and when measured is sometimes indistinguishable from sensor data.

4.35. (Sensor) Platform
An entity to which can be attached sensors or other platforms. A platform has an associated local coordinate reference frame that can be referenced relative to an external coordinate reference frame and to which the reference frames of attached sensors and platforms can be referenced.

4.36. Transducer
An entity that receives a signal as input and generates a modified signal as output. Includes detectors, actuators, and filters.

4.37. Value
A member of the value-space of a datatype. A value may use one of a variety of scales including nominal, ordinal, ratio and interval, spatial and temporal. Primitive datatypes may be combined to form aggregate datatypes with aggregate values, including vectors, tensors and images [ISO11404].

5 Conventions

5.1 Abbreviated terms

In this document the following abbreviations and acronyms are used or introduced:

CRS	Coordinate Reference System
DN	Digital Number
ECEF	Earth-Centered Earth-Fixed
ECI	Earth Centered Inertial
GPS	Global Positioning System
ISO	International Organization for Standardization
MISB	Motion Imagery Standards Board
OGC	Open Geospatial Consortium
SAS	Sensor Alert Service
SensorML	Sensor Model Language
SI	Système International (International System of Units)
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
TAI	Temps Atomique International (International Atomic Time)
uom	Unit(s) of measure
UCUM	Unified Code for Units of Measure
UML	Unified Modeling Language
UTC	Coordinated Universal Time
XML	eXtended Markup Language
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional

5.2 UML notation

The diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram. The UML notations used in this standard are described in the diagram below.

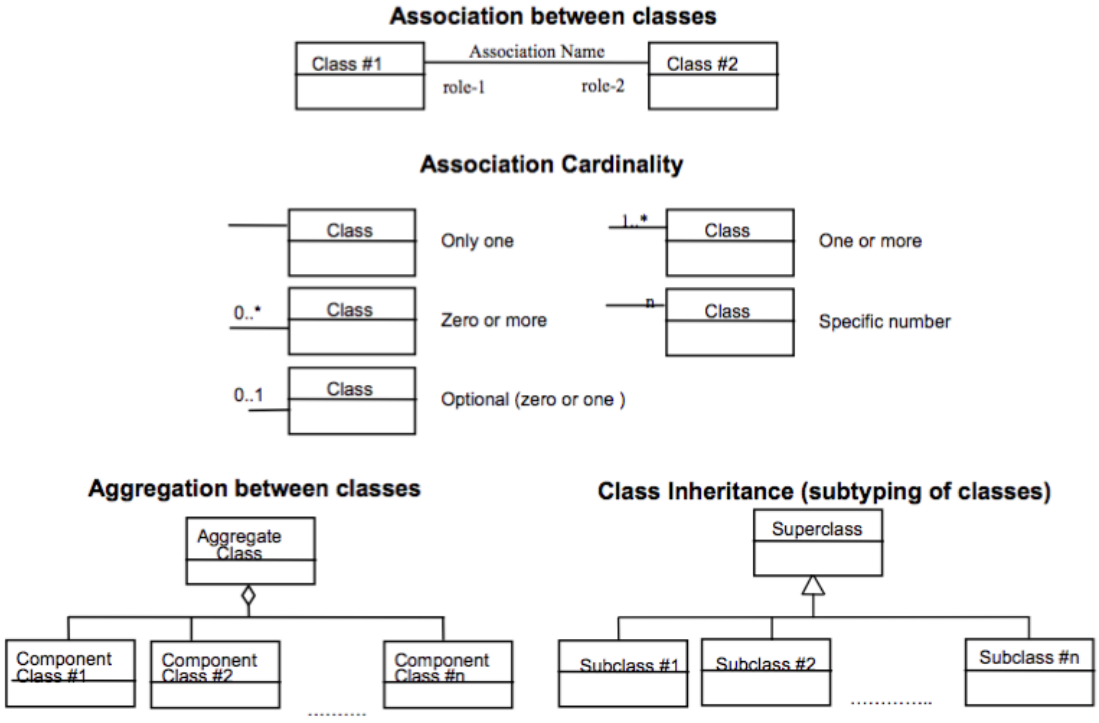


Figure 5.1 – UML Notation

5.3 Table notation used to express requirements

For clarity, each normative statement in this standard is in one and only one place and is set in a **bold font** within the tabular format shown below. If the statement of the requirement is repeated for clarification, the "bold font" home of the statement is considered the official statement of the normative requirement. Individual requirements are clearly highlighted and identified throughout the document by using tables and URL identifiers of the following format:

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/{req-class-name}/{req-name}
Req N. Textual description of requirement.

In this standard, all requirements are associated to tests in the abstract test suite in Annex A. The reference to the requirement in the test case is done by its URL.

Requirements classes are separated into their own clauses and named, and specified according to inheritance (direct dependencies). The Conformance test classes in the test suite are similarly named to establish an explicit and mnemonic link between requirements classes and conformance test classes. There are formally identified by URL and described within a tabular format as shown below:

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/{req-class-name}	
Target Type	Description of standardization target type
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/{req-class-name}

Examples will be indicated using a grey text box and are considered to be informative rather than normative:

Example
This is an example text box that will include informative notes and examples.

6 Requirements Class: Core Concepts (normative core)

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/core	
Target Type	Derived Model, Encoding, and Software Implementation

6.1 Introduction

In SensorML, all components are modeled as processes. This includes components normally viewed as hardware, such as detectors, actuators, and physical processors (which are viewed as physical components) and sensors and platforms (which are viewed as physical systems). All components are modeled as processes that receive input and through the application of an algorithm defined by a method and set parameter values, generate output. All such components can therefore participate in process networks (or aggregate processes). Aggregate processes are themselves processes with their own inputs, outputs, and parameters.
Hence, SensorML can be viewed as a specialized process description language with an emphasis on application to sensor data. Process descriptions in SensorML are agnostic of the environment in which they might be executed, or the protocol by which data is exchanged between process execution modules.

In order to support the use of SensorML within specialized applications (e.g. processing centers or image processing software), the SensorML models and encodings have been divided into several conformance classes. Thus if one wishes to use SensorML for computation processes only, the software only needs to conform to the requirements for non-physical processes. Similarly, by only adhering to the Simple Process conformance class, a piece of software can describe internal processes using SensorML while supporting chaining of these processes in a proprietary way.

However, all derived model and encodings based on SensorML shall implement the core concepts of SensorML, regardless of whether they deal strictly with non-physical computational processes or sensor systems.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/core/core-concepts-used
Req 1. Any derived model or encoding shall correctly implement the modeling concepts defined in the core of this specification.

6.2 Process Definitions

In SensorML, all relevant components are modeled as processes, including both computation and physical processes (e.g. detectors, actuators, and sensor systems). Processes in SensorML are conceptually divided into two types: (1) those that are physical processes, such as detectors, actuators, and sensor systems, where information regarding their positions may be relevant, and (2) non-physical or "pure" processes which can be treated as merely mathematical operations or functions.

Fundamentally, a process is a physical or computational operation that may receive input and based on configurable parameters and a methodology, generate output.

Example

For a process representing the standard linear equation, x would be the input, m and b the parameters, y the output, and the equation y = mx + b would define the methodology.

For a detector, the input would typically be a physical stimulus (or observable property), the parameters might include a calibration curve and other factors that affect the measurement, and the output would be a digital number representing some quantity representation of that observed property.

Inputs and outputs may be digital numbers or physical stimuli (i.e. observable properties of the environment). Parameters can be variable or constant, but they don't typically vary at the same frequency as the input values. In essence, however, parameters can be viewed as just another input into the process that is either fixed or changes less frequently than inputs

A process can consist of a single atomic operation, or an explicitly defined network of operations (e.g. an aggregate process or system).

Any process shall have a definable method of operation. In the case of an aggregate process or physical system, the explicit description of the process components and the flow of data between them will itself serve as the process methodology.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/core/processes
Req 2. The core model for a process shall define inputs, outputs, parameters, and methodology of that process.

Any process description shall provide a unique ID that can be used for discovery of that process and for retrieving the definition of that process.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/core/unique-id
Req 3. The core model for a process shall include a unique ID for distinguishing that process from all others.

To be useful, the core process model shall include metadata about the process that aid in identification, discovery, and qualification of the process but do not themselves affect the execution of the process.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/core/metadata
Req 4. The core model for a process shall include metadata that support identification, discovery, and qualification of the process.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/core/execution
Req 5. The metadata descriptions for a process shall not be required for successful execution of that process. All information required for execution of a simple process shall be contained within the inputs, outputs, parameters, and methodology descriptions of the process.

Process definitions can support general representations of a process or a specific instance of a process.

Example
A general process for the linear equation would define the allowable inputs, outputs, and parameters. A specific instance of the process might define constant values for the parameters.
An example of a general physical process would be the manufacturer's description of the characteristics and configurable options for a particular model of a sensor (i.e. one that describes the common characteristics of all instances of that model of sensor). The description of a specific instance of that model of sensor would include information that is relevant to that particular instance of the sensor (e.g. serial number, owner's name, location, etc.).

7 UML Conceptual Models (normative)

This standard defines normative UML models with which derived encoding models as well as all future separate extensions should be compliant. The standardization target type for the UML requirements classes defined in this clause is thus a software implementation or an encoding model that directly implements the conceptual models defined in this standard.

7.1 Package Dependencies

The packages defined by the SensorML Model and their dependencies are shown in figure below:

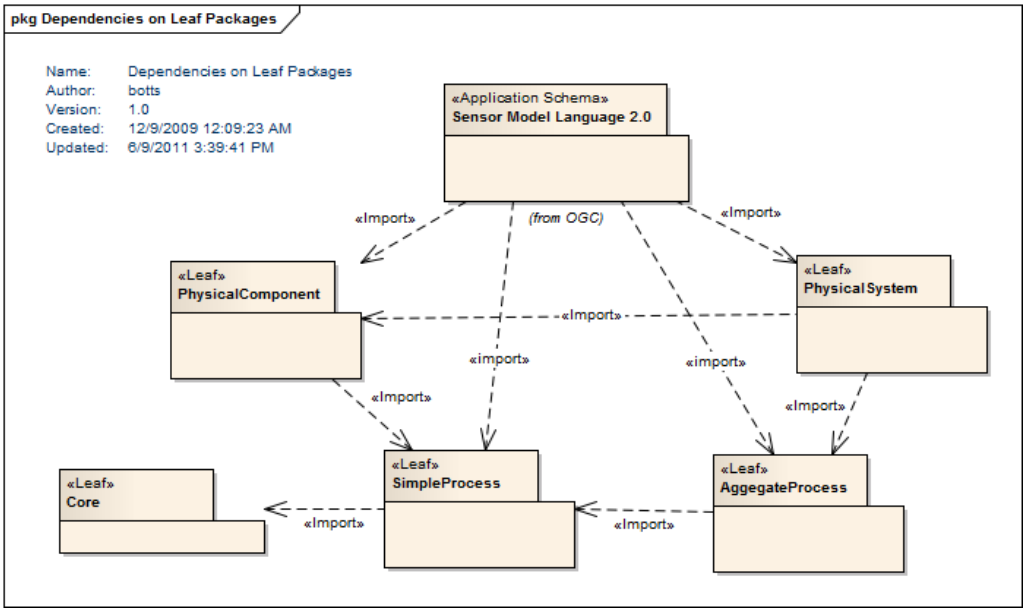


Figure 7.1 – Internal Package Dependencies

SensorML also has dependencies on several external packages defined within other standards, namely GML 3.2, ISO 19103, ISO 19108, ISO 19111, and ISO 19115, as described below.

7.1.1 Dependency on GML Feature Model and ISO TC 211 Models

A process represents a feature type as defined by the ISO General Feature Model and is thus modeled as an instance of the <<metaclass>> *GF_FeatureType* (ISO 19109:2006). These models are further refined within the GML 3.2 standard (ISO 19136) which will provide the basis for XML encodings in the following section 8 of this specification

document. This association of SensorML process with GML primarily brings recognition of SensorML processes as features and provides important identity properties.

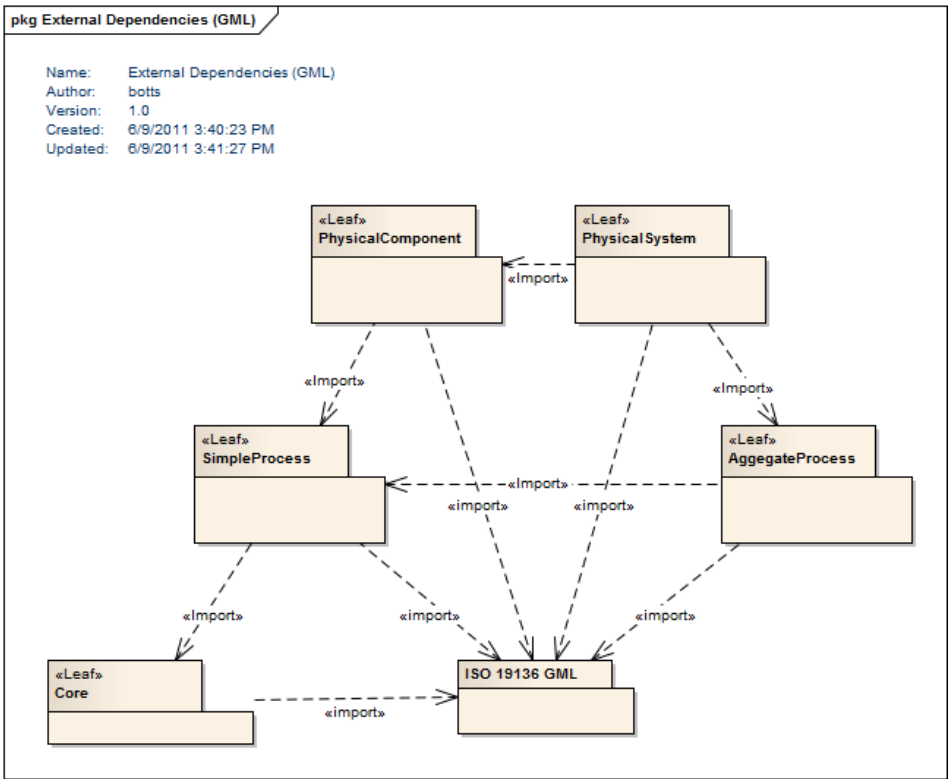


Figure 7.2 – External Package Dependencies – GML

The base feature classes in GML from which all processes in SensorML derive include AbstractGML and AbstractFeature, as shown in the figure below:

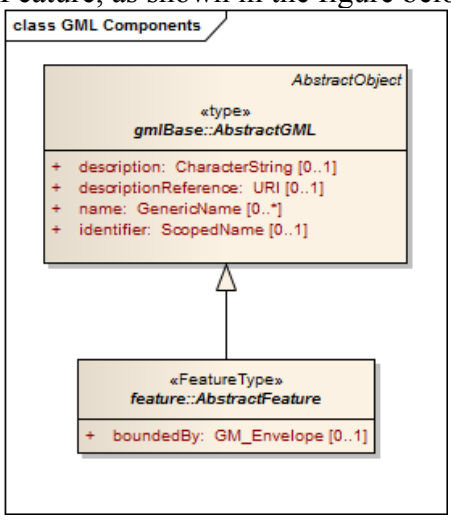


Figure 7.3 – Models for dependent GML Feature classes

SensorML is dependent on ISO 19108:206 for Temporal Schema. In particular, the temporal elements, *TM_Instant* and *TM_Period* are used within the core SensorML model for Abstract Process. Additionally, SensorML depends on ISO 19115 for general metadata elements.

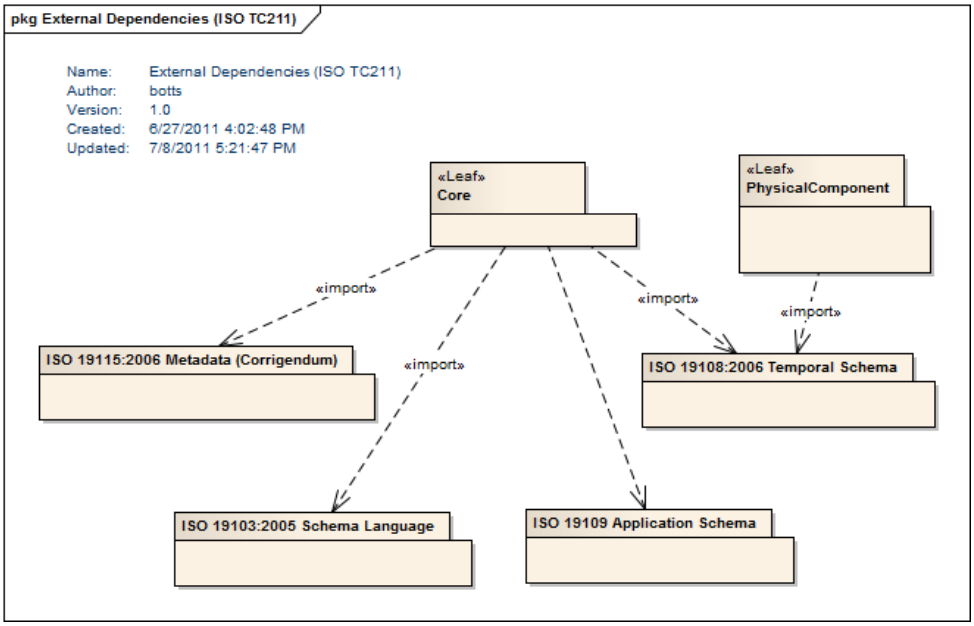


Figure 7.4 – External Package Dependencies – ISO TC 211

The SensorML standard utilizes the ISO 19115 models for common metadata properties such as keywords, citations, online resources, responsible party, and constraints. While Version 1.0 of SensorML defined encoding based on the ISO 19115 models, this version utilizes these models directly.

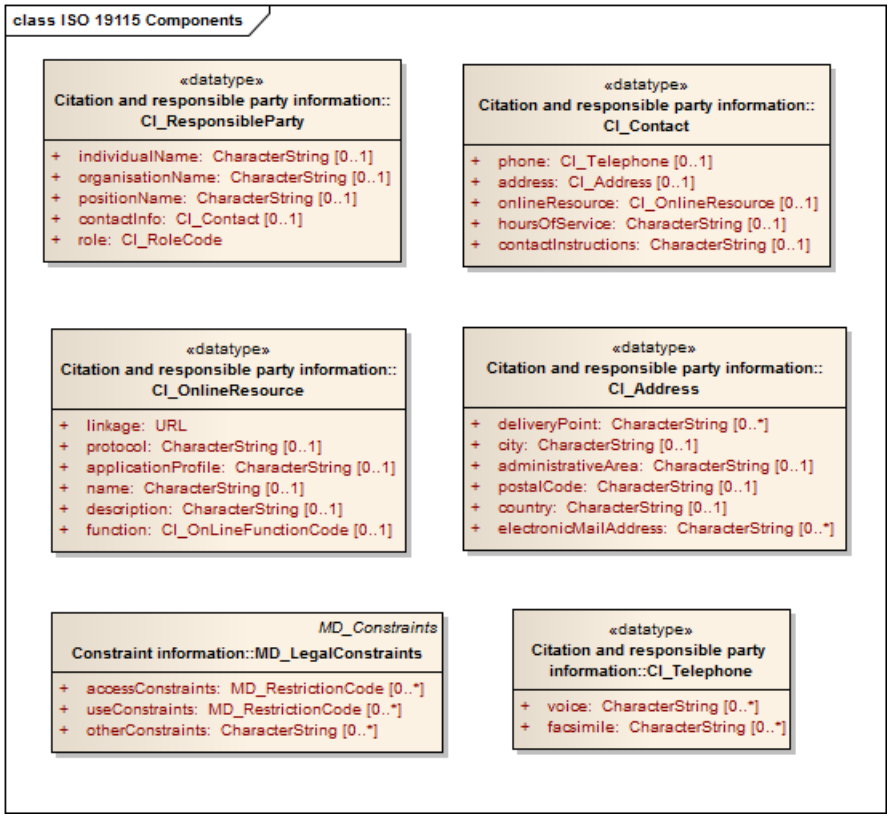


Figure 7.5 – ISO 19115 Models for dependent classes.

7.1.2 Dependency on SWE Common Data Models

In particular, SensorML is heavily dependent on the SWE Common Data Model standard for defining inputs, outputs, and parameters, as well as for specifying characteristics, capabilities, interfaces, and event properties. The SWE Common Data Models, which were originally defined within the version 1.0 SensorML specification, are in version 2.0 defined as a separate specification and are utilized throughout the SWE family of encoding and web service specifications.

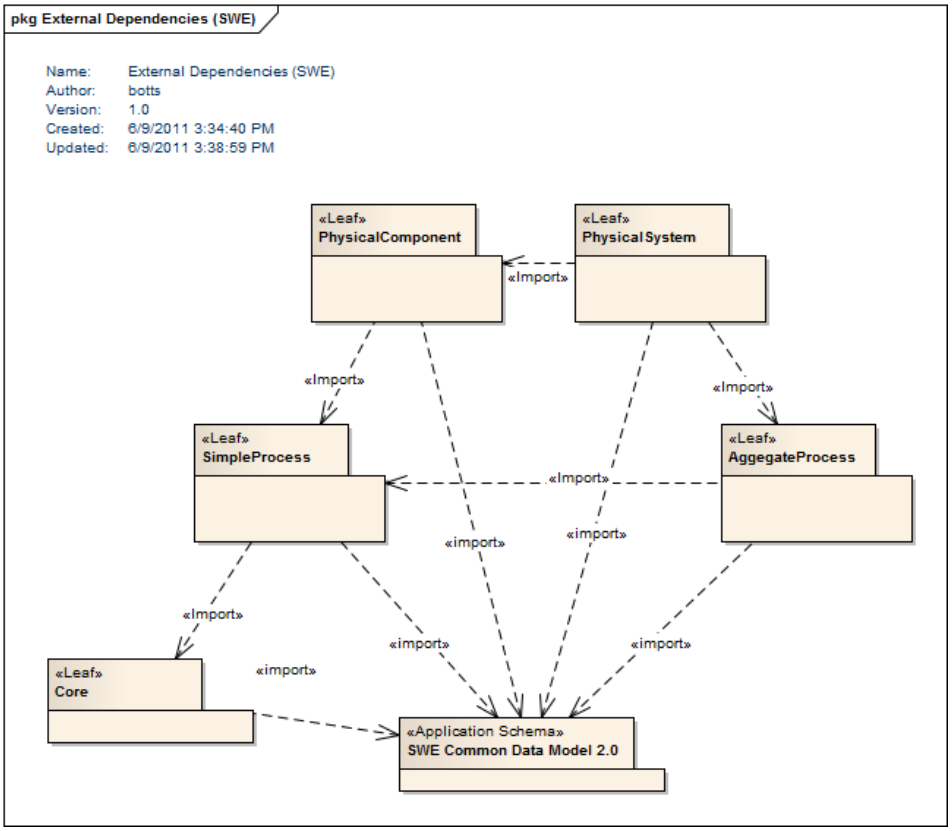


Figure 7.6 – External Package Dependencies – SWE Common Data

The SWE Common specification provides a flexible yet robust means of defining data types and data values, including support for simple data types such as *Quantity*, *Boolean*, *Category*, *Count*, *Text*, and *Time*, as well as aggregate data such as *DataRecord*, *DataArray*, *Vector*, and *Matrix*. Additionally, SWE Common supports the concept of *DataChoice,* which will be utilized by SensorML for providing multiplexed messages in data streams and configurable options for processes and physical systems.

The data models in SWE Common provide additional properties than are provided by basic data types, including for example, units of measure (uom), quality indications, allowable constraints, significant digit counts, and in particular, the meaning and semantics of a data component. Both simple and aggregate data components in SWE Common allow for unambiguous definition of that data component through a resolvable link to an online dictionary or ontology. The definition of the SWE Common Data Models can be found in OGC 08-094r1.

The main objective of SWE Common Data Models is to achieve interoperability, first at the syntactic level, and later at the semantic level (by using ontologies and semantic mediation) so that sensor data can be better understood by machines, processed automatically in complex workflows, and easily shared between intelligent sensor web nodes.

SensorML depends heavily on the *AbstractDataComponent* element defined in SWE Common. This element serves as the base component from which all relevant data types in SWE Common are derived, including *Quantity*, *Count*, *Category*, *Boolean*, *Text*, *DataRecord*, *DataArray*, *Vector*, *Matrix*, and *DataChoice*. *AbstractDataComponent* thus serves as a substitution group that any of these data types can satisfy. *AbstractSWEIdentifiable* will serve as the basis for the *ObservableProperty* element defined in this specification (Section 7.2.1).
The model for the SWE Common *AbstractDataComponent* is given in the figure below:

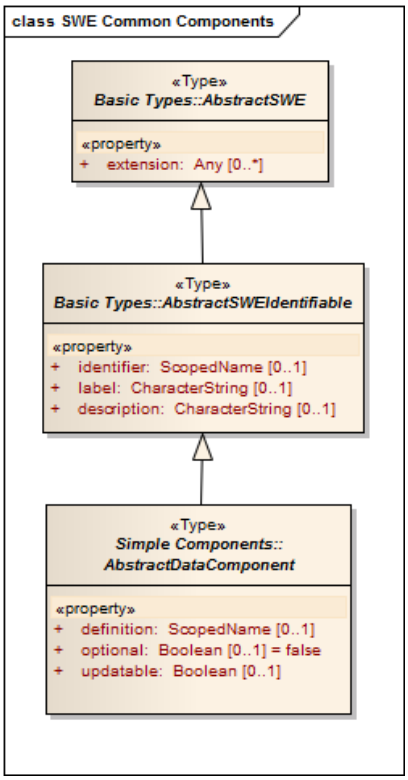


Figure 7.7 –Models for dependent SWE Common AbstractDataComponent class.

7.1.3 Relationship to Observations and Measurements (O&M)

Conceptual models for Observations and Measurements are provided by ISO 19156, which also provides models for sampling feature types. XML Schema encodings of these models are provided by the OGC Observations and Measurements XML Implementation Document (OGC 10-025). The model for *Observation* defines a procedure of type *AbstractFeature* which references or describes the origin of the observation (i.e. how the observation came to be).
SensorML has an association to the O&M models but no direct dependencies on them. The result of a SensorML process is typically considered to be an observation result if it

is measuring or deriving some value of a physical property or phenomenon. Thus, the *output* values described in SensorML and resulting from a sensor or process may be packaged in an O&M *Observation* object, or provided as a SWE Common *DataStream*. Inversely, the *procedure* property within an *Observation* instance may reference a SensorML description of the measurement process.

7.2 Requirements Class: Core Abstract Process

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process	
Target Type	Derived Encodings and Schema
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/core
Dependency	OGC 08-094r1 (SWE Common Data – uml-block-components)
Dependency	ISO 19115:2006 (All Metadata)
Dependency	ISO 19136 (GML)

All major classes in SensorML are based on a process model, as presented in the core concepts. Processes are features as defined in ISO 19109:2006 and modeled in GML 3.2. SensorML also supports interoperable discovery, identification, and qualification of these processes through the definition of a standard collection of metadata.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/dependency-core
Req 6. A schema or encoding definition passing the "Core Abstract Process" model conformance class shall first pass the "Core Concepts" conformance test class.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/package-fully-implemented
Req 7. A schema or encoding definition shall correctly implement all UML classes defined in this section

7.2.1 ObservableProperty

An *ObservableProperty* is a physical property of a phenomenon that can be observed and measured (e.g. temperature, gravitational force, position, chemical concentration, orientation, number-of-individuals, physical switch status, etc.), or a characteristic of one or more feature types, the value for which will be estimated by application of some procedure in an observation. It is thus a physical stimulus that can be sensed by a detector or created by an actuator.

Example
The ObservableProperty element allows one to reference a measurable property of a phenomenon or feature for detector inputs or actuator outputs. For example, the temperature of the atmosphere is an ObservableProperty. Before measurement, it is simply a property of the atmosphere that can be defined and measured. After measurement by a detector, the temperature may be represented as a Quantity with units of measure, a value, and an indication of our degree of confidence in the measurement.

ObservableProperty is derived as a concrete instance of the SWE Common *AbstractSWEIdentifiable* and adds the *definition* property to this model. It will be used as a potential input (e.g. for detectors), output (e.g. for actuators), and for parameters (e.g. for a sensor whose measurement varies with fluctuations of atmospheric pressure on a diaphragm).
In *ObservableProperty* the phenomenon property will be defined by reference using the *definition* attribute. The *definition* attribute value will reference a property defined within a dictionary or ontology. An *ObservableProperty* may also include a name and a description. However, unlike the simple data types in SWE Common, an *ObservableProperty* does NOT include the properties uom, quality, or constraints, since these are typically characteristics of the measuring procedure and not properties of the observable phenomenon itself.

7.2.2 DescribedObject

As shown in the UML model below, the *DescribedObject* class provides a specific set of metadata for all process classes in SensorML. Since *DescribedObject* is itself derived from GML *AbstractFeature*, all processes in SensorML are themselves features, which conforms to the conceptual models for processes as stated in Section 6.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/gml-dependency
Req 8. *DescribedObject* shall derive from the GML base class, *AbstractFeature*, and is thus modeled as a feature with well-defined metadata properties. Any model or encoding derived from *DescribedObject* shall thus be of type *featureType*.

The GML *AbstractFeature* inheritance provides a unique ID, and support for multiple names and a description. The unique ID in SensorML will be supported by a single *gml:identifier* property inherited from GML *AbstractFeature*.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/unique-id
Req 9. A single, required *gml:identifier* property inherited from GML *AbstractFeature* shall be used to provide a unique ID for the *DescribedObject*.

Metadata about each process is essential to supporting identification, discovery, and qualification of the process. Metadata is provided by the base class, *DescribedObject*, from which *AbstractProcess* is derived. While these metadata may provide relevant information to understand quality of output from the process, the values of properties within the *DescribedObject* should not be required for execution of the process. The model for the *DescribedObject* is shown in Figure 7.8., while the models for the individual property values are provided in either Figure 7.9 or in the ISO 19115 models in Figure 7.5.

The DescribedObject includes several descriptive properties that support rapid discovery (*keywords*, *identification*, and *classification*), constraints (*validTime*, *securityConstraints*, *legalConstraints*), qualification (*characteristics* and *capabilities*), references (*contacts* and *documentation*), and *history*. These are each grouped in lists, which provide for easy seaparation and parsing of these properties.

7.2.2.1 Extension Property

The *extension* property allows one to add domain or community-specific content to a *DescribedObject* instance. This might include, for example, security taggings, vendor or community-specific metadata, or information encoded in other models or schema. Extension properties exist in a separate namespace and SensorML-compliant software is not required to understand or utilize the information contained within the extension property.

The constraints on the *extension* property include: (a) the extension model shall be defined in a separate namespace, (b) the information added by the extension model shall not be required for execution of the process, and (c) SensorML-compliant parsers may parse and utilize the information within these extensions but they are not required to do so in order to be compliant to the SensorML standard.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/extension-independence
Req 10. Models inside of the extension property shall exist within a namespace other than SensorML.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/extension-restrictions
Req 11. Information provided inside of the extension property shall not be required for execution of the process and shall not alter the execution of the process.

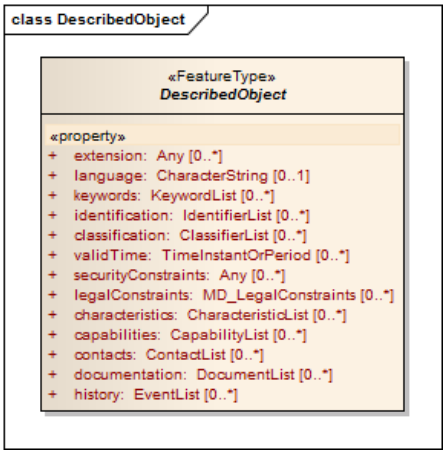


Figure 7.8 – DescribedObject with Metadata Properties

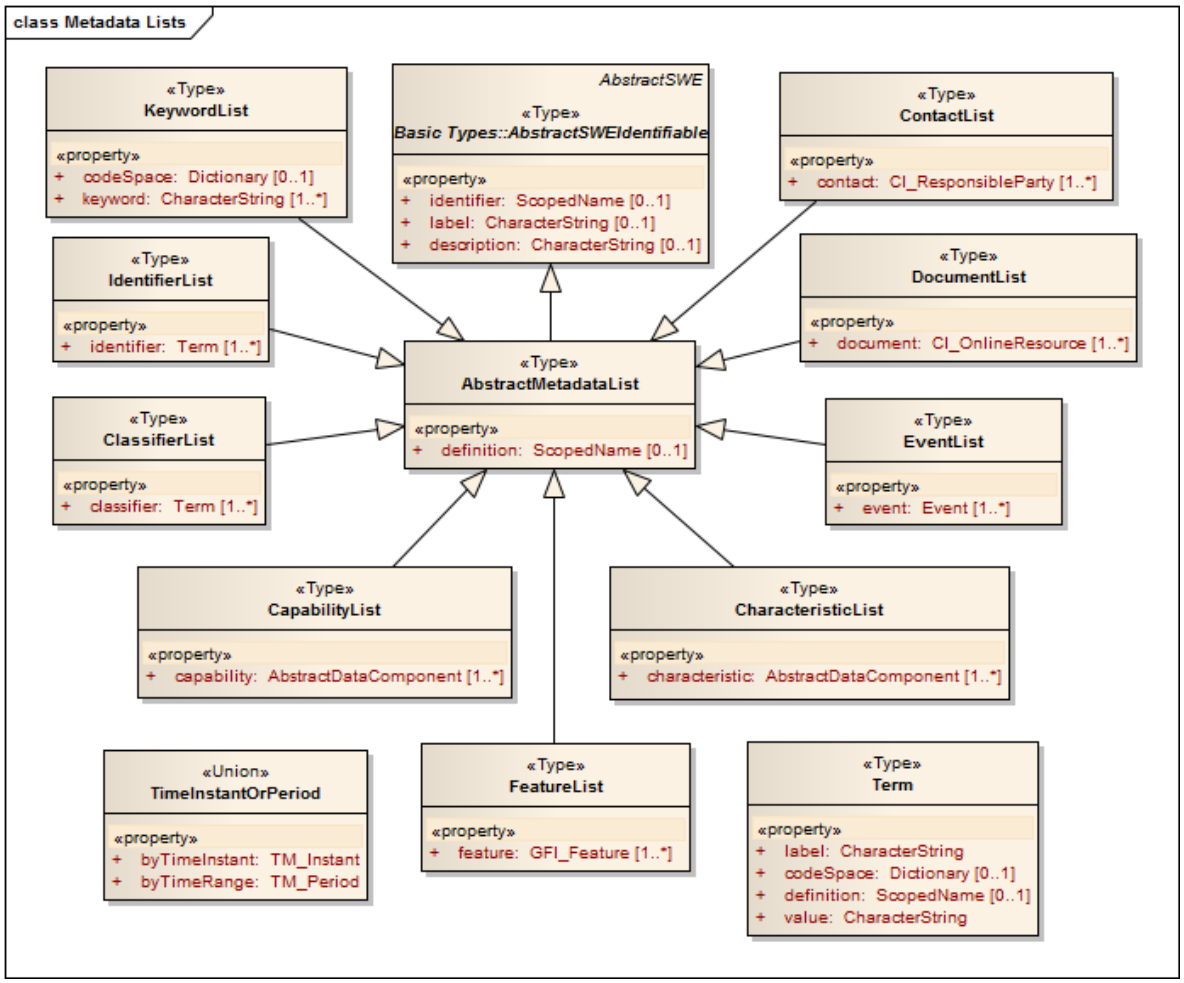


Figure 7.9 – Models for Metadata Elements

7.2.2.2 Keywords

Keywords provide a simple means of discovery using short tokens that may be recognized by the general audience or specific communities. Keywords are unqualified terms in that they are not necessarily required to be related to a specific codespace or ontology, as are classifiers and identifiers.

7.2.2.3 Identifiers

The *identifier* property takes a *Term* as its value. The *Term* has a *definition* attribute that specifies in this case the type of identifier, while the *codeSpace* attribute specifies that the value of the identifier is according to the rules or enumerations of a particular authority. The *identification* properties should be considered as information suitable for the discovery applications.

Example
An identifier with a definition of "http://sensors.ws/def/tailNumber" might take "N291PV" as its value based on the codespace of a US Air Force rules dictionary. Other possible definitions for identifiers might include, for example, shortName, longName, acronym, missionID, processorID, serialNumber, manufacturerID, or partNumber.

7.2.2.4 Classifiers

The classifier property provides a list of possible classifiers that might aid in the rapid discovery or organization of processes, sensors, or sensor systems. The *classifier* properties should be considered as information suitable for the discovery and categorization applications.

Example
Definitions for a classifier Term might include, for instance, sensorType, observableType, processType, intendedApplication, or missionType.

7.2.2.5 Security Constraints

The model for specification of security constraints will be based on external security models. The *securityConstraints* property takes a value of *xs:Any* which allows various communities and countries to utilize their standard XML encoding for security tags. This security constraint is for the overall document. As will be discussed in the XML encoding, extension points provided with SWE Common Data elements will allow security tagging for individual properties or property aggregates.

Example
One can specify the overall security classification of the entire document using the Intelligence Community Information Security Banner Marking (IC ISM) standard or using ISO 19115 MD_SecurityConstraints. For tagging individual sections in the document, the SensorML standard allows for security tagging of properties using an extension property, as describe in later sections of the standard.

7.2.2.6 Valid Time Constraint

The *validTime* property indicates the time instance or time range over which this process description is valid. Time constraints are important for processes in which parameter values or operation modes may change with time, or instrument deployment times change.

Example
Several SensorML documents can exist for the same sensor or system description but with different validity periods. This allows for capturing the configuration of a sensor at different times and, along with the history section, is the basis for maintaining history of the sensor's description. Alternately, parameter values can be provided as a time-tagged series of values accounting for changes.

7.2.2.7 Legal Constraint

The *legalConstraints* property is based on ISO 19115 and specifies whether such legal and ethical considerations as privacy acts, intellectual property rights, copyrights, or scientific publication ethics apply to the content of the process description and its use.

7.2.2.8 Capabilities

The *capabilities* property is intended for the definition of properties that further qualify the input or output of the process, component, or system for the purpose of discovery. These properties are defined using one or more SWE Common *DataRecord* elements.
Once a user has identified candidate sensors or processes based on the classifiers described above, the capabilities parameters might prove useful for further filtering of processes or sensor system during this discovery stage. Thus, the *capabilities* properties should be considered as information suitable for the discovery process.

Example
A particular remote sensor on a satellite might measure radiation between a certain spectral range (e.g. 700 to 900 nanometers) at a particular ground resolution (e.g. 5 meter), and with a typical spatial repeat period (e.g. 3.25 – 4.3 days). Alternatively, a particular process might have certain quality constraints. Any process may have certain limits (e.g., operational and survivable limits), based on physical or mathematical conditions.
These properties do affect the output of the process and should be considered as capabilities.

7.2.2.9 Characteristics

A physical or non-physical process may have *characteristics* that may not directly qualify the output. These properties are defined using one or more SWE Common *DataRecord* elements.

Example
A component may have certain physical measurements such as dimensions and weight, and be constructed of a particular material. A component may have particular power demands, or anticipated lifetime.These are characteristics of the component that may not directly affect the output of the component or system.

The *characteristics* properties may or may not be considered as information suitable for the discovery process.

7.2.2.10 Contacts

Contact information can provide access to manufacturers, system experts, equipment owners, or any other persons responsible in some way for design, deployment, maintenance, or additional information regarding the *DescribedObject*. The *contact* property within the *ContactList* takes the ISO 19115 classes *CI_ResponsibleParty* as its values.

7.2.2.11 Documentation

Documentation can be provided which provides further clarification about the *DescribedObject*. This might include technical manuals, manufacturer brochures, journal references, or theoretical-basis documents. The *DocumentList document* property takes the ISO 19115 *CI_OnlineResource* as it value.

7.2.2.12 History

Within SensorML, the history of a process can be provided through a collection of *Event* objects. These are provided within an *EventList* that serves as the value of the *history* property. Events might for instance, specify calibration or maintenance history of a sensor, changes to an algorithm or parameter within a computational process, or deployment and maintenance events.

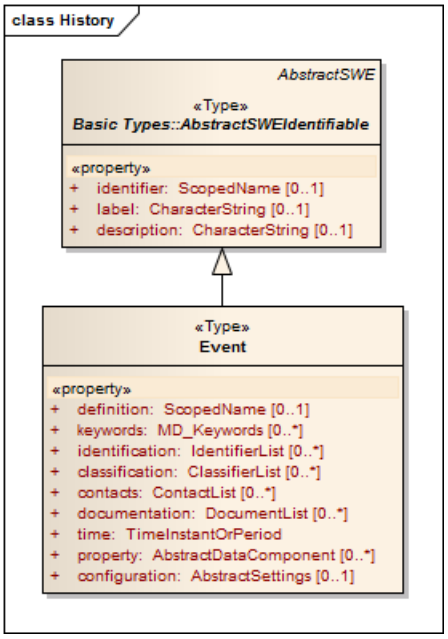


Figure 7.10 – Model for history events

7.2.3 AbstractProcess

As discussed in the Core Concepts, the major elements of SensorML are modeled as physical and non-physical processes. All SensorML process elements will derive from *AbstractProcess,* shown in Figure 7.11. The class *AbstractProcess* itself derives from the *DescribedObject* class and thus inherits a wide range of optional metadata supporting discovery, identification, and qualification and an option for domain and community-specific extensions. In addition to the metadata provided by *DescribedObject*, the *AbstractProcess* includes the properties of *inputs*, *outputs*, and *parameters*, as required by the process model defined in the Core Concepts, as well as the properties *typeOf*, *featureOfInterest*, *configuration*, and *modes* which will be discussed below.

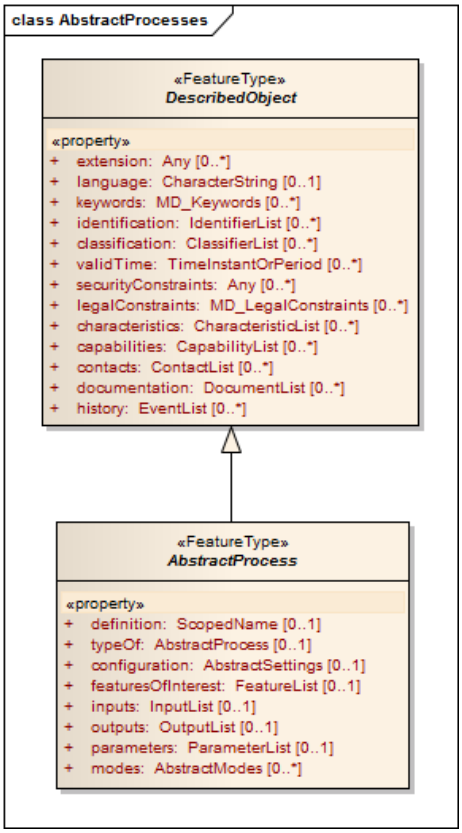


Figure 7.11 – UML models for DescribedObject and AbstractProcess

7.2.3.1 Inputs, Outputs, and Parameters

As discussed in the Core Concepts, any process can have *inputs*, *outputs*, and *parameters*. Processes typically receive input and based on the parameter settings and methodology, generate output. Some processes, such as detectors, receive physical stimulus as input and generate digital numbers as output. In such cases, the input would be represented as an *ObservableProperty*, and the output as a *DataComponent* (e.g. a *Quantity*). If this output is encoded and accessible directly, then the output can be represented as a *DataInterface*.

Example
A digital thermometer is stimulated by an observable property of the environment (temperature), which is modelled as its input (ObservableProperty), and outputs a digital number (Quantity) that represents a measure of that property.

Thus, an *AbstractProcess* model supports the *inputs*, *outputs*, and *parameters* properties in conformance with the Core Concepts. These properties can accept *ObservableProperty* or SWE Common elements *AbstractDataComponent* or *DataStream* as their values. Classes derived from *AbstractDataComponent* include *Quantity*, *Count*, *Category*, *Boolean*, *Text*, and *Time*, as well as ranges and aggregates of these simple data types.

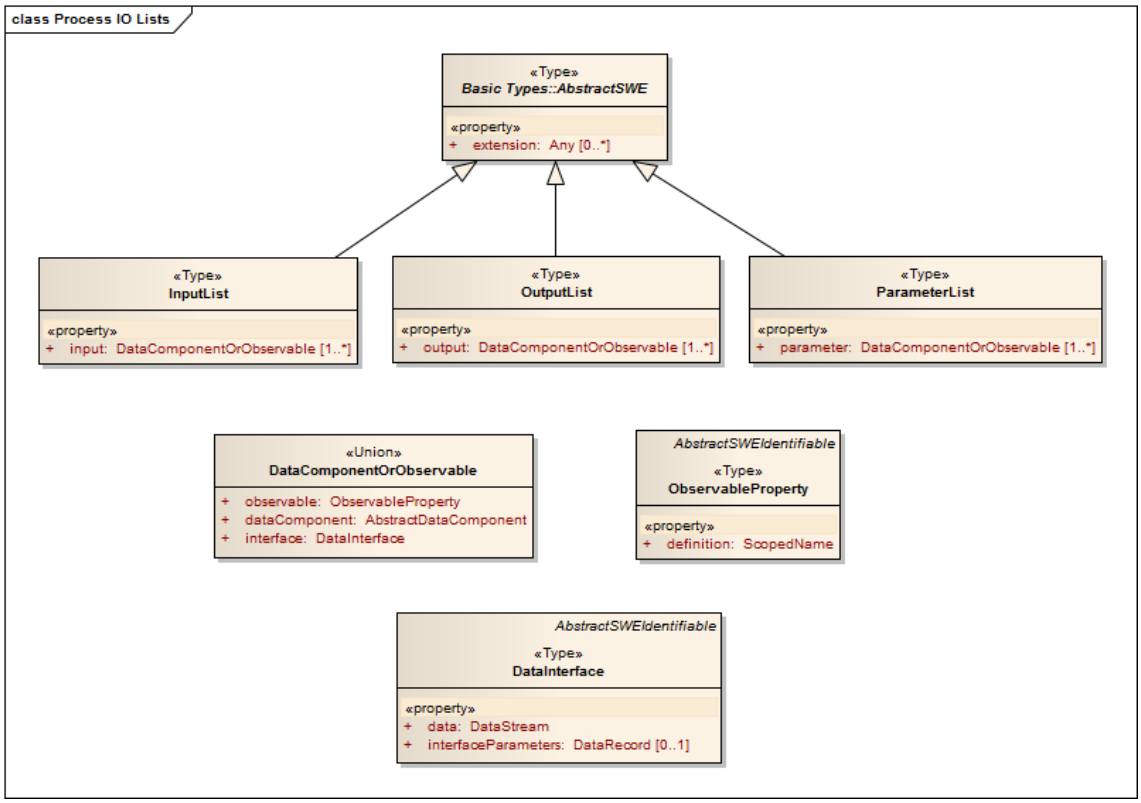


Figure 7.12 – UML models for process inputs, outputs, and parameters.

The core process model will utilize the SWE Common Data Models for defining inputs, outputs, and parameters, as well as for other metadata properties. SensorML models are required to support the SWE Common Data Model up to the Block Components Requirements Class, but many instances of SensorML will find ALL conformance levels of SWE Common Data to be useful, including binary encodings.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/swe-common-dependency
Req 12. Any derived model or encoding for process shall utilize *ObservableProperty* or SWE Common Data Components as values for inputs, outputs, and parameters,

and shall at a minimum conform to the SWE Common Data "Block Components Package" class (http://www.opengis.net/spec/SWE/2.0/req/uml-block-components).

The input, output, or parameters of many processes include multiple values, possibly of different data types, that are tightly related to one another. Sometimes referred to as tuples or records, these data aggregates can consist of values that are perhaps meaningless without the other associated values (e.g. the coordinates within a spatial reference system), or provide a more complete understanding because of their association with one another (e.g. a set of measured values taken by a sensor at a given time). Such data will be modelled using the aggregate data types defined by the SWE Common Data standard.

Examples
The location of a dynamic object can be specified through the aggregate values of time, latitude, longitude, and altitude. In such cases, the expression of one of the values separate from the others is meaningless or less complete than the expression of these values as a set or aggregate. These four values should be encapsulated in a Vector data type that also identifies the reference frame in which the latitude, longitude and altitude coordinates are expressed

Weather stations often express a set of measurements of the atmosphere as a single record that might include for instance temperature, pressure, relative humidity, cloudiness, wind speed, and wind direction. These would be considered a tuple of values that provides a more complete picture of the environment at a particular time. This tuple should be modeled as a DataRecord with 7 fields (one for each measured parameters listed above + one time stamp) to indicate that the sampling time applies to all observable values included in the record.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/aggregate-data
Req 13. Multiple input, output, and parameter values that are tightly related with one another shall be modeled as a SWE Common Data aggregate data type.

7.2.3.2 Feature of Interest

Most sensors and many non-physical processes have been deployed or implemented with a focus on one or more features of interest. Within SensorML, the primary purpose of including a *FeatureOfInterest* property for *AbstractProcess* is to support discovery as well as to further clarify the intended purpose of the physical or non-physical process.

Example
The features of interest of an installed web camera might include a particular building, a particular street, or a general area of observation surrounding the camera. Features of interest for other sensors might include the Gulf of Mexico, a particular drilling well, the atmosphere surrounding a particular weather station, a particular patient, or a particular automobile. Features of interest for a model or other process might include a particular river basin, a particular toxic plume release, or a particular metropolitan area.

7.2.3.3 Inheritance, Extension, and Configuration

SensorML supports the concepts of inheritance, extension, and configuration. In other words, generalized base processes can be described in SensorML and then that description can be augmented or further constrained by one or more separate descriptions. Thus, a single, generalized description of a physical or non-physical process can serve as a basis for one or many more specific process descriptions. This provides support for more simple and concise process descriptions while also providing the ability for the user or application to "drill down" to greater and greater detail as desired.

The inheritance model will support two cases:

a) Simple inheritance – the specific process description provides only additional information to the description of the general process, without modifying or restricting any property values of the general process.

b) Configuration – the specific process description is able to set or restrict property values within the allowable range provided by the general process description, as well as provide additional information.

The key to inheritance, extension, and configuration of a process lies in the *typeOf* property, by which a specific process can reference its more general base process. The *typeOf* property takes as its value an instance of any process model derived from *AbstractProcess*. This will be "by-reference-only", meaning that the value will be in the form of a resolvable link to another process instance.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/type-of
Req 14. A process that is a specific instance of another process shall reference the more general process through its *typeOf* property. The value of the *typeOf* property shall be a resolvable link to an instance of a process derived from *AbstractProcess*.

7.2.3.3.1 Simple Inheritance

In the simple inheritance model, a process (referred to as the "specific process") inherits and augments information from another process (referred to as the "general process").

Example
An Original Equipment Manufacturer (OEM) provides a description of a particular model of their sensor that would define inputs, outputs, and parameters, as well as perhaps capabilities, characteristics, manufacturer contact information and documentation relevant to that model. Thousands of sensors of this model type may of course be manufactured and sold by the OEM.
When one purchases and deploys an instance of that model of sensor, the owner can then reference the OEM's description of the model and provide additional information that's specific to his specific instance of the sensor. Additional information might include, for example, serial number, owner's contact information, the sensor's location, calibration data, and the interface description for accessing the data.

The simple inheritance model is fully supported in the Core Process conformance class and will be supported solely through the use of the *typeOf* property within the specific process. The *typeOf* property within the specific process will reference the general process through a resolvable reference.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/simple-inheritance
Req 15. A process instance that references another process through the *typeOf* property, but does not include the *configuration* property, shall inherit properties of the referenced process through simple inheritance. The complete description of that process is thus the addition of information from both process descriptions.

7.2.3.3.2 Support for Configurable Processes

A configurable process is one that includes options or choices that can be selected, restricted, or enabled during deployment, operation, or execution of that process.

Example
An Original Equipment Manufacturer (OEM) can provide a description of a particular model of their sensor that would define inputs, outputs, and parameters, as well as perhaps capabilities, characteristics, manufacturer contact information and documentation relevant to that model. In addition, the OEM enables an individual instance of that model of sensor to be configured by providing options for setting parameter values, setting modes, or choosing a particular interface. Thousands of sensors of this model type may of course be manufactured and sold by the OEM.
When one purchases and deploys an instance of that model of sensor, the owner can then reference the OEM's description of the model and provide additional information that's specific to that particular instance of the sensor. In addition, the owner can configure the sensor by setting values, selecting modes, and enabling particular interfaces. These settings would be provided in the instance description.

The configuration model will utilize both the *typeOf* and *configuration* properties. The *typeOf* property references the more general process as with simple inheritance, while the *configuration* property provides a means to further restrict the options and allowed values

for the specific process. The configuration property in the *AbstractProcess* takes an *AbstractSettings* class as its value.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/configuration
Req 16. A process instance that references another process through the *typeOf* property, and further restricts options or allowed values provided in the referenced process, shall specify those restrictions through the *configuration* property.

A concrete implementation of a *Settings* class will be provided in a later Conformance Clause.

7.2.4 SWE Common Data Types

Many properties in the *DescribedObject* and *AbstractProcess* classes described above are of type *AbstractDataComponent* as defined in the SWE Common Data Model standard. This data type is used for defining inputs, outputs and parameters, as well as for other metadata properties.

This requirements class only mandates the support of the "Simple Components" and "Record Components" as defined in the SWE Common Data Model standard. These includes the scalar data types Boolean, Text, Count, Quantity, Category, Time and their range equivalents, as well as DataRecord and Vector.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/swe-common-dependency
Req 17. Contents of all properties of type *AbstractDataComponents* shall pass the SWE Common Data Model "Records Components Package" conformance test class.

However, many implementations of SensorML will find ALL conformance levels of the SWE Common Data Model to be useful, including arrays, choices and encodings. An implementation claiming support for more than the record components can pass the "Processes with Advanced Data Types" conformance test class of this standard.

7.3 Requirements Class: Simple Process

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process	
Target Type	Derived Encodings and Schema
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/core-process

A simple process is derived from abstract process model, as presented in Clause 7.2.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/dependency-core
Req 18. A schema or encoding passing the "Simple Process" model conformance class shall first pass the "Abstract Process" requirements test class.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/package-fully-implemented
Req 19. A schema or encoding definition shall correctly implement all UML classes defined in the "SimpleProcess" package described in this section.

7.3.1 Simple Process Definition

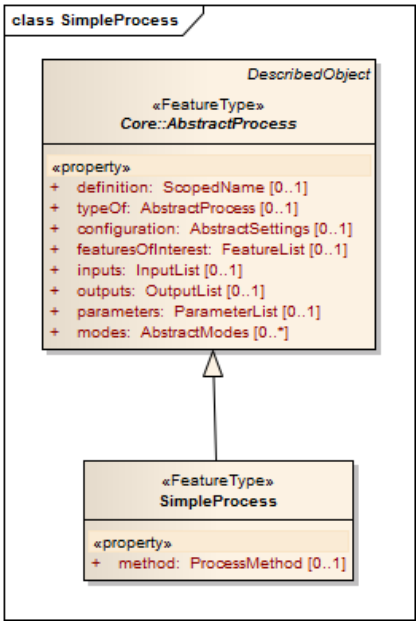
A simple process is a process that, for whatever reason, is considered indivisible. That is, there is no intent to further divide the process description into an aggregation of sub-processes. While the process method may describe several steps within the algorithm, the actual execution of this process is expected to occur as a single modular unit.
Often simple processes are computational processes that can be executed with an associated piece of software. Simple processes are often one component of a physical or non-physical aggregate process.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/definition
Req 20. A process shall be modeled a "Simple Process" if it provides a processing function with well-defined inputs and outputs, if there is no intent to further

divide the process description into sub-process components, and if knowledge of its physical location is of no importance.

The *SimpleProcess* model, as shown in Figure 7.13, is a concrete instantiation of the *AbstractProcess* model. The *SimpleProcess* requires a method description.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/method
Req 21. A schema or encoding of the SimpleProcess class shall support the definition of the method.



Example
A process computing a simple mathematical function such as sine, cosine or square root is usually modeled as a SimpleProcess instance. However, even more complex processes can be modeled this way if there is no intent to break down the implementation of the process into sub-processes.

Figure 7.13 – Model for Simple Process

7.3.2 Process Method Definition

The *ProcessMethod* provides a description of the methodology used by the process to execute and generate output based on the input and parameter values. This includes a textual description, as well as an optional description of the algorithm in an appropriate format (e.g. mathML) and optional references to particular executable implementations.

The *ProcessMethod* definition should be sufficient to allow one to understand how input values are converted to output values, given a particular set of parameter values, and be able to write software that is capable of executing this process.

A *ProcessMethod* description may be protected by security or legal constraints, which would purposely prevent access to the method description as well as restrict knowledge of the methodology to authorized personnel only. However, regardless of access restrictions, a *ProcessMethod* should always be able to be referenced and identified by a unique identifier.

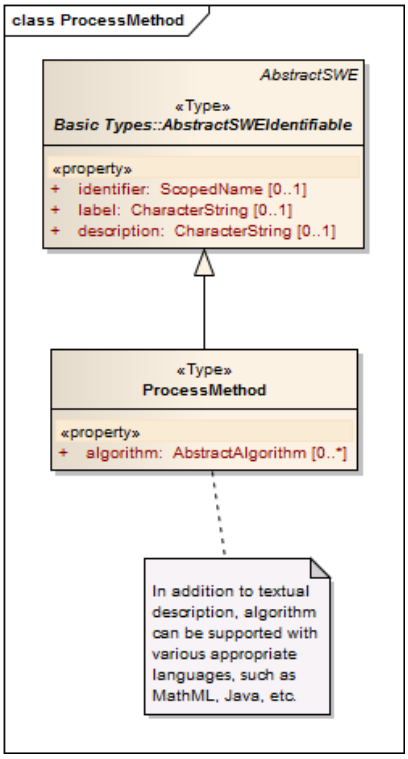


Figure 7.14 – Model for ProcessMethod

7.4 Requirements Class: Aggregate Process

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process	
Target Type	Derived Encoding or Software Implementation
Dependency	http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process

An aggregate process is derived from abstract process model, as presented in Clause 7.2.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/dependency-core
Req 22. A schema or encoding definition passing the "Aggregate Process" conformance test class shall first pass the "Simple Process" conformance test class.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/package-fully-implemented
Req 23. A schema or encoding definition for an aggregate process shall correctly implement all UML classes defined in the "AggregateProcess" package described in this section.

7.4.1 Aggregate Process Definition

An aggregate process is a collection of autonomous component processes with an explicit mapping of the data flow between these processes. Components of an aggregate process can be simple processes (i.e. atomic) or be aggregate process themselves. Aggregate processes can include both physical and non-physical (i.e. logical) components.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/definition
Req 24. A process shall be modeled as an aggregate process if it provides a processing function with well-defined inputs and outputs, if there is intent to further divide the process description into sub-processes, and if knowledge of its physical location is of no importance.

Requirement

http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/components
Req 25. A schema or encoding implementation of the AggregateProcess class shall support the inclusion of one or more component processes and a means for explicitly specifying data flow between these components.

In SensorML, an aggregate process is agnostic to the execution engine that may perform the actual execution of individual sub-processes and manage the execution sequencing and the flow of data between the components. Also, while it is possible in SensorML to more explicitly define the data encoding if desired by using the encoding specifications defined in the SWE Common Data Specification, SensorML is typically agnostic to the protocol and format of data flowing between logical processes.

This provides significant flexibility as to where and how a SensorML-defined aggregate process is executed. While the *ProcessMethod* explicitly defines the algorithm for executing an atomic process, the actual execution of that algorithm and the management of data flow between processes can be handled by any software system able to parse a SensorML-defined aggregate process and sequence the execution of the processes.

A SensorML-defined process component or aggregate process can be executed through web services, within the CPU of a laptop, mobile device, or supercomputer, or a mix of these. Furthermore, a SensorML-defined aggregate process can be executed wherever desired, be it at a large data or computation center, within a visualization and analysis client on a laptop, or on-board a sensor or actuator system. Thus, SensorML provides the choice to either bring the process to the data or bring the data to the process.

The model for *AggregateProcess* is shown in the figure below. *AggregateProcess* extends the *AbstractProcess* model and adds one or more process components and explicit linking of data flow between these components. The *component* property takes any component derived from *AbstractProcess* as its value. Component process descriptions can be provided inline or by reference.

The derivation from *AbstractProcess* means that an *AggregateProcess* instance itself has its own inputs, outputs, and parameters, as well as identification and possible metadata.

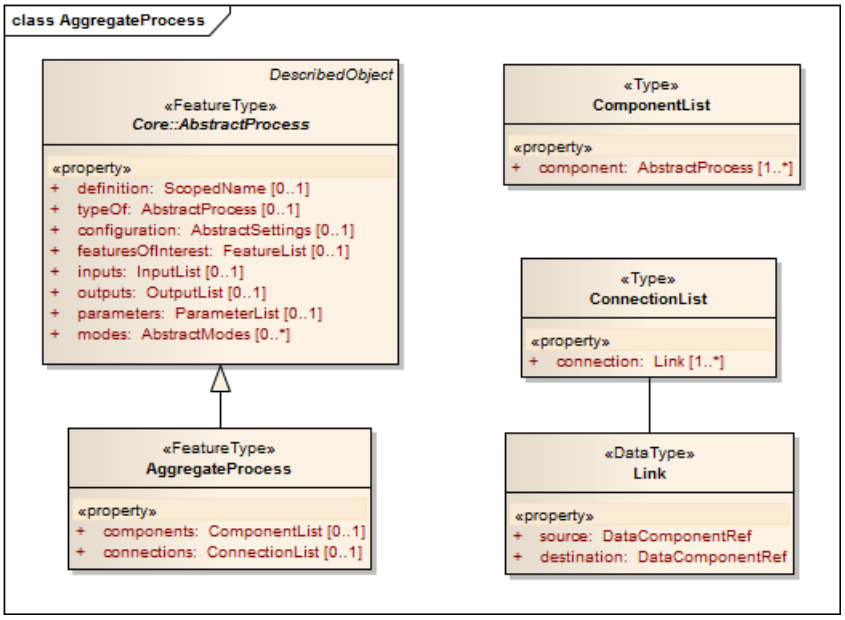


Figure 7.15 – Model for Aggregate Process

7.6 Requirement Class: Physical Component

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component	
Target Type	Derived Encodings and Schema
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/core-process

In the context of SensorML, physical processes represent real processing devices whose spatio-temporal position is important. Physical processes include detectors, actuators, sensor systems, and actuator systems. Such processes typically, but not always, involve interactions beween a real-world domain (or environment) and a digital domain.

Example
A detector or sensor system typically senses an environmental stimulus and provides a digital number representing the measure of a property of that environment (e.g. temperature). Likewise, an actuator receives a digital number and based on its values causes an action in the real-world environment.Both devices interact with the real world and their position is usually of importance to the end-user. These should usually be modelled as physical components

Because physical processes typically interact with the real-world environment, the position (location and orientation), as well as perhaps the dynamic state (velocity and acceleration), are usually of importance. We wish to either measure an observable property at a particular location in the environment or we wish to affect a physical action at a particular place in the environment. Thus, the position where the physical process measures or acts becomes important.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/package-fully-implemented
Req 26. A schema or encoding definition shall correctly implement all UML classes defined in the "PhysicalComponent" package described in this section.

7.6.1 Abstract Physical Process Defined

The *AbstractPhysicalProcess* model is derived from *AbstractProcess* and thus includes the metadata and properties of a core process. Additionally, *AbstractPhysicalProcess* supports additional properties that allow one to define spatial and temporal coordinates for the physical process device.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/dependency-core
Req 27. A schema or encoding passing the "Physical Component" conformance class shall first pass the "Core Abstract Process" conformance test class.

The model for *AbstractPhysicalProcess* is shown in Figure 7.16 below. The additional properties of the *AbstractPhysicalProcess* will be discussed in subsequent clauses.

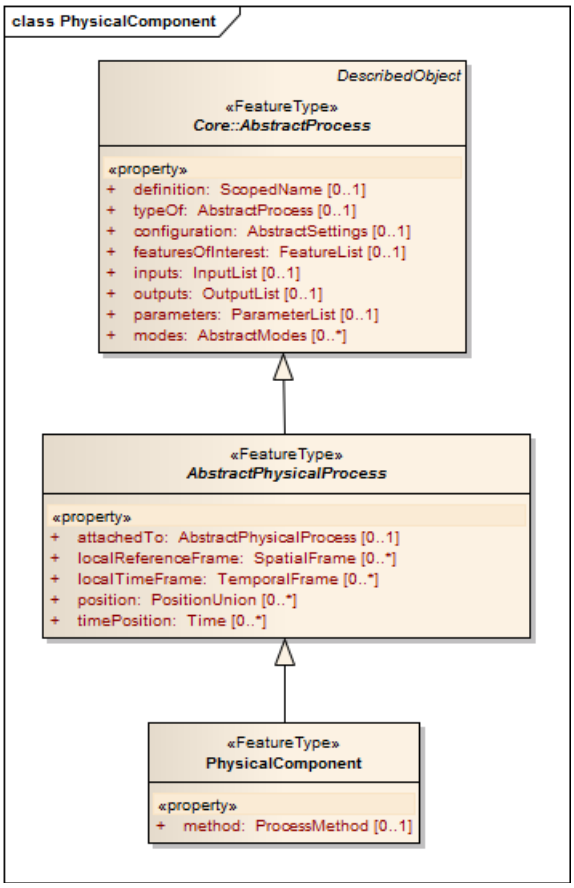


Figure 7.16 – Model for Physical Process Component

7.6.1.1 attachedTo Property

A physical process ("child process") may be attached to another physical process ("parent process") such that the movement of the parent process affects the position of the child process. The *attachedTo* property provides a reference from the attached process to the process to which it is attached.

Example:
A video camera is attached to a gimbal that allows rotation of the camera to view a 360° area surrounding the camera. In such a case, the camera is said to be attached to the gimbal. Both are physical processes (the camera, a sensor; the gimbal, an actuator). The video camera description should thus use the attachedTo property to reference the gimbal description.

7.6.1.2 Position and Spatial Reference Frames

In this standard, the position or dynamic state of a physical object is defined as a relationship of the reference frame of the object to some external reference frame. SensorML allows for the definition of direct orthogonal (i.e. Cartesian) reference frames that are assumed to be attached to the physical component where they are described.

A reference frame is defined by its origin and its axes, which are described relative to the physical object itself using natural language and are not relative to any relationship of the object to some external frame. The relationship of this object's reference frame to an external reference frame is defined by the position or dynamic state of the object. The models for reference frames and spatial position are provided in Figure 7.17.

Example:
The origin of an airplane's spatial reference frame can be defined as the being at the center of the Inertial Navigation Unit main gyro. The axes can be defined by the following statements: "X is along the symmetric axis of the airplane's fuselage from the gyro to the nose of the airplane (along the platform roll axis of the airplane), Z is perpendicular to X and toward the belly of the airplane (along the yaw axis of the aircraft), and Y is Z cross X (in the direction of of the right wing and along the pitch axis of the airplane)".
The location of this aircraft can then be given as the spatial relationship of the origin of this reference frame to some external reference frame (e.g. Earth-Centered-Earth-Fixed XYZ or latitude-longitude-altitude). Likewise, the orientation of the aircraft can be specified as the angular relationship of the axes of its reference frame to the axes of some external reference frame (e.g. ECEF or North-East-Down).

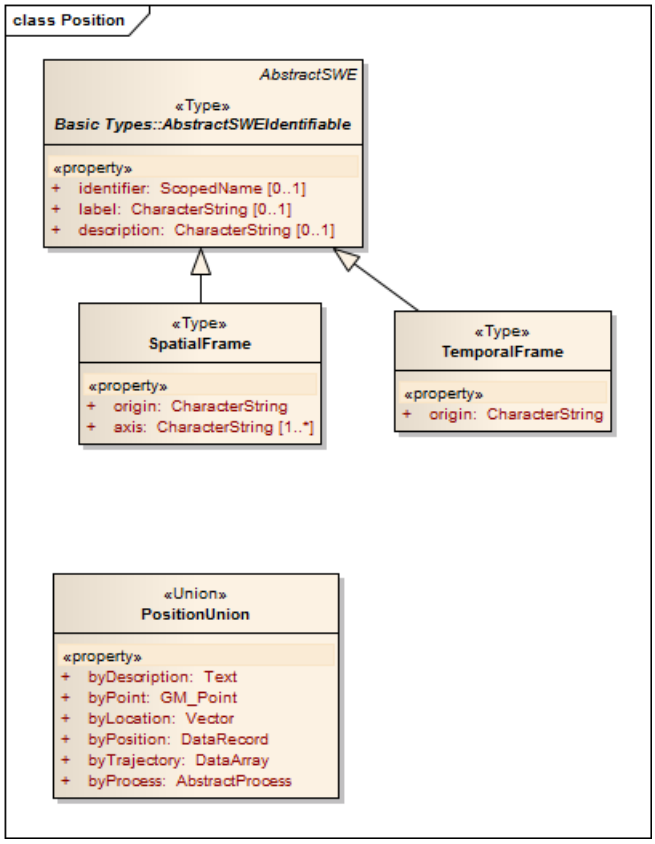


Figure 7.17 – Models for SpatialFrame and PositionUnion

In this standard, "position" is defined as the combination of location and orientation. Location is the linear displacement (translation) of the origin of the object's spatial reference frame relative to the origin of some external reference frame (which will be designated). The orientation of an object is the angular relationship between the axes of the object's reference axes to those of some external reference frame. The dynamic state of an object can include its time-tagged location, orientation, linear velocity, angular velocity, and higher-order derivatives when required (e.g. linear and angular acceleration, jerk, etc.).

An external reference frame can be another object's reference frame (e.g. the reference frame of a ship) or a geographic reference frame (e.g. WGS84 latitude-longitude-altitude).

The *PositionUnion* class provides various means of specifying the location, position, or dynamic state of an object. These will be described in more detail in the appropriate XML encoding section, but the following rules apply to the SensorML models.

Requirement

http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-point-or-location
Req 28. Specification of position "byPoint" or "byLocation" shall specify the location of the origin of the object's reference frame relative to the origin of a well-defined and specified external reference frame.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-position
Req 29. Specification of position "byPosition" shall specify, using two Vectors, the location and orientation of the object's reference frame relative to a well-defined and specified external reference frame.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-trajectory
Req 30. Specification of position "byTrajectory" shall specify, at a minimum, the time-tagged location of the object's reference frame relative to a well-defined and specified external reference frame, but may also include its orientation and any number of derivatives of the location and orientation.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-process
Req 31. Specification of position "byProcess" shall specify SensorML-modeled process whose output provides, at a minimum, the time-tagged location of the object's reference frame relative to a well-defined and specified external reference frame, but may also include its orientation and any number of derivatives of the location and orientation.

7.6.1.3 Temporal Reference Frames

Just as spatial position must be related to a spatial reference frame, time must also be related to a temporal reference frame. Temporal reference frames can include a particular calendar, a particular time of day reference frame, or a frame attached to an event of interest.

Example:
A temporal frame can be attached to an event of interest, such as the start of the mission. When such a reference frame is defined, time measurements can be expressed in seconds past the mission start time (which is usually itself referenced to a global time frame such as UTC or TAI).

A temporal reference frame can be defined within a physical process and is particularly useful if the component is a process that outputs its own measure of time (such as an on-board clock or high-resolution counter).

7.6.2 Physical Component Defined

Any processing device can be considered a physical component, if it provides a processing function with well-defined inputs and outputs, if there is no intent to further divide the device description into component sub-processes, and if knowledge of its physical location is useful. Such devices could include, but not be limited to, detectors, actuators, reflectors, electrical components (e.g transformers, capacitors, resistors), or perhaps even computational units (when knowing their location in a computational facility is helpful).

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/definition
Req 32. A process shall be modeled as a "Physical Component" if it provides a processing function with well-defined inputs and outputs, if there is no intent to further divide the device description into sub-process components, and if knowledge of its physical location is of importance.

As shown in the models of Figure 7.16, the *PhysicalComponent* class is a concrete instantiation of an *AbstractPhysicalProcess* that adds the *method* property, which takes a *ProcessMethod* as its value. *ProcessMethod* was defined earlier in clause 7.3.2.

7.7 Requirement Class: Physical System

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system	
Target Type	Derived Encodings and Schema
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/physical-component

A physical system is used to model a hardware device as an aggregate process made of one or more components and whose location in the real world is known and of importance.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system/package-fully-implemented
Req 33. A schema or encoding definition supporting physical systems shall correctly implement all UML classes defined in the "PhysicalSystem" package described in this section.

Sensor and actuator systems (e.g. machines and robots) are typically physical systems that perform a particular feat through the coordinated actions of both physical and non-physical sub-processes. Even though a sensor system's overall application is to sense something in the environment, the system itself can consist of sensing components (e.g. detectors and sensing subsystems), action (e.g. actuators and robotic subsystems), and computational components.

Example:
A weather station is an example of physical system that is composed of several sensors (thermometer, barometer, wind sensor, etc.) and other computational process such as an algorithm to compute wind chill. All these components can be described in SensorML and grouped in a PhysicalComponent description representing the station as a whole.
A hand-held digital camera can also be modeled as a physical system with an overall task of sensing radiance in a scene and generating an image. However, the camera is an aggregate of various sub-processes, each of which can be physical or non-physical, and can be sensing, acting, or computational. For example, a light detector outputs a measure of brightness, which serves as the input of a computational process which outputs a signal that provides input into an actuator that controls the opening or closing of the iris. The final iris size is sensed by another detector which inputs that value into a process that encodes that value into an EXIF format that accompanies the image, which is generated by a entirely different subsystem of the camera.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system/definition

Req 34. A process shall be modeled as a "Physical System" if it provides a processing function with well-defined inputs and outputs, if the device description is further divided into subprocess components, and if knowledge of its physical location is of importance.

The model for *PhysicalSystem*, as shown in Fig. 7.18, is derived from *AbstractPhysicalProcess*, and adds the *components* and *connections* properties that have been described in the non-physical counterpart, *AggregateProcess* (Clause 7.4).

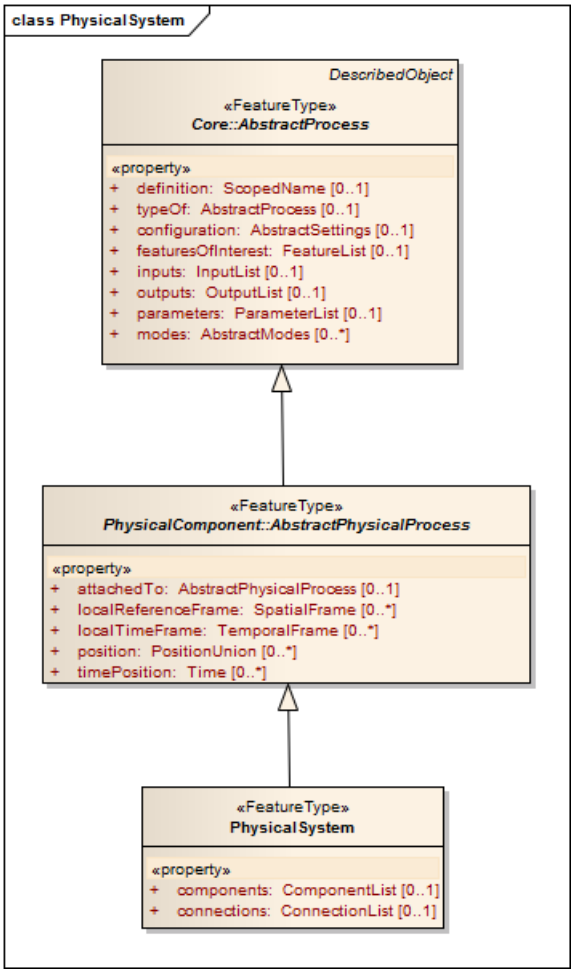


Figure 7.18 – Model for Physical Processing System

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system/dependency-core
Req 35. A schema or encoding passing the "Physical System" model conformance test class shall first pass the "Physical Component" conformance test class.

7.8 Requirements Class: Processes with Advanced Data Types

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/model/advanced-process	
Target Type	Derived Encodings and Schema
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/core-process
Dependency	OGC 08-094r1 (SWE Common Data – uml-block-components)
Dependency	OGC 08-094r1 (SWE Common Data – uml-choice-components)

The "Core Abstract Process" requirements class only requires the support of the record and scalar data types wherever a data type from the SWE Common standard is used.
This class also requires support for more advanced data types defined in the SWE Common standard : DataArray, Matrix, DataStream and Choice.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/advanced-process/dependency-core
Req 36. A schema or encoding definition passing the "Advanced Data Types" model conformance class shall first pass the "Abstract Core Process" conformance test class.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/advanced-process/package-fully-implemented
Req 37. A schema or encoding definition shall correctly implement all UML classes defined in this section.

7.9 Requirements Class: Configurable Processes

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process	
Target Type	Derived Encodings and Schema
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/core-process

Many processes, both physical and non-physical, are configurable in that they provide one with the ability to set parameters values, enable options, or select modes before or during execution/operation. Thus a general configurable process can be defined and published specifying allowed values for parameters, modes that can be selected, and options that can be enabled or disabled.
A specific process that inherits from this general process can then refine the process in several ways by: (1) specifying values for parameters, (2) further constraining the allowable values of parameters, (3) selecting an operational mode (which then sets a group of parameter values), or (4) enabling or disabling particular options such as particular outputs or components.
In this document, we will refer to the more general process as the "configurable process", and the more specific process that inherits from it, as the "configured process".

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/dependency-core
Req 38. A schema or encoding definition passing the "Configurable Process" conformance test class shall first pass the "Core Abstract Process" conformance test class.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/package-fully-implemented
Req 39. A schema or encoding definition shall correctly implement all UML classes defined in the "Configuration" package described in this section.

A process will be considered "configurable" if it provides options, variable parameters, or modes that can be selected or set before or during deployment or execution.
A process will be considered "configured" if it sets options, parameter values, or modes that were defined within the configurable process.

Example:
A configurable process based on the linear equation (y=mx+b) defines two parameters for "slope" and "y-intercept" but does not provide values for these parameters. A configured process can inherit from this configurable process and set the values of those parameters (e.g. y=2x+4).

A process becomes "configurable" by one or more of the following characteristics:

a) it defines *parameters*, but not defining their values
b) it defines a range or selection of possible values for *parameters* using the *swe:AllowedValues* property
c) it defines *modes* which in turn set a collection of parameter values when enabled
d) it allows *inputs*, *outputs*, or *components* to be enabled or disabled

A process becomes "configured" by having both of the following two characteristics:

a) it inherits from a configurable process using the *typeOf* property
b) it specifies one or more settings within the *configuration* property

7.9.1 Modes

A process mode is a specific configuration in which the values for multiple parameters can be set and multiple options selected by enabling that mode. Often modes are established as a convenience so that a process can be configured during deployment or execution in order to best meet some specific operational need.

Example:
A Doppler radar for monitoring storms may have several modes from which one can select depending on the prevailing conditions at the time. For instance, there can be "clear-sky", "storm", and "severe-storm" modes in which the scanning properties, radar intensity, and gain settings can all change by simply changing the mode setting

A configurable process can but is not required to contain one or more *modes* properties. The *modes* property takes an *AbstractModes* as its value. As shown in Figure 7.19, the concrete *ModeChoice* class defined in this conformance clause is derived from *AbstractModes* and serves as the concrete instantiation for defining modes in this specification.
ModeChoice will include two or more *mode* properties that take *Mode* as their value. In addition to metadata provided by the base *DescribedObject* class, *Mode* includes a *configuration* property that allows one to define a collection of settings for that mode.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/two-modes-required
Req 40. A *ModeChoice* instantiation shall include two or more *mode* properties.

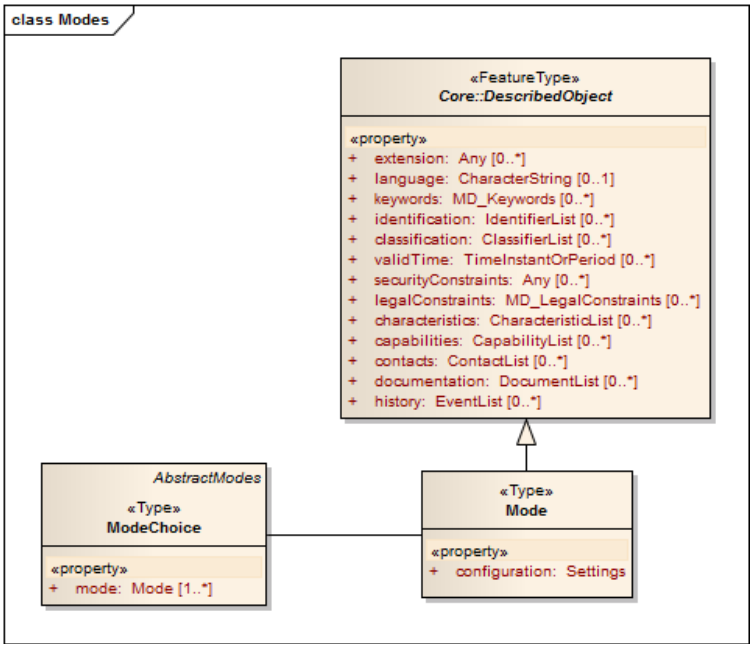


Figure 7.19 – Model for Modes

The *configuration* property takes a *Settings* object, which will be described in more detail below.

7.9.2 Settings

The *configuration* property and it *Settings* value can be utilized in two cases:

a) within the *Mode* definition of a configurable process for defining a collection of settings for that particular mode
b) as a required property within a configured process for setting one or more configurable properties

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/settings-property
Req 41. A configured process shall include a *configuration* property that takes a *Settings* class as its value.

The *Settings* class is shown in Fig. 7.20 with its possible property values shown in Fig. 7.21. For all settings, the property in the configurable process is specified by the *DataComponentPath* reference.
Within the *Settings* class, one may (a) set particular values for parameters, (b) set an array of values for a parameter (and only a parameter that takes a *DataArray* as its value, (c)

further constrain allowed values for parameters, (d) set the operational mode, and (e) enable or disable an input or output.

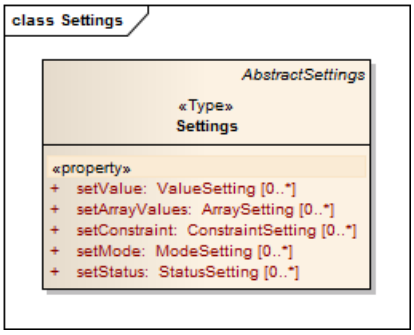


Figure 7.20 – Model for Configured Process Settings

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/set-value-restriction
Req 42. The *setValue* property shall only reference and set values for a *parameter* defined in a configurable process.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/set-array-value-restriction
Req 43. The *setArrayValues* property shall only reference and set array values for a *parameter* defined in a configurable process.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/set-constraint-restriction
Req 44. The *setConstraint* property shall only reference and set constraints for a *parameter* defined in a configurable process.

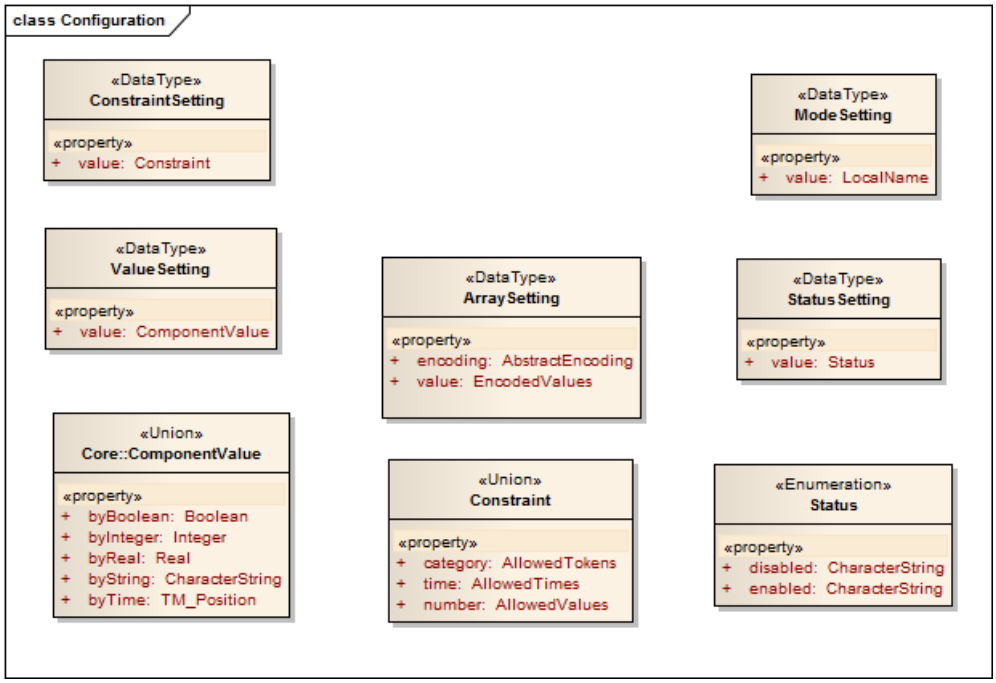


Figure 7.21 – Model for Settings Elements

8 XML Schema Implementation (normative)

This standard defines a normative XML Schema grammar and a set of patterns that represent an XML implementation of the conceptual models presented in Section 7. The standardization target type for all requirements classes in this clause is an XML instance that seeks compliance with this XML encoding model.

XML schemas defined in this section are a direct implementation of the UML conceptual models described in Section 7. They have been automatically generated from these models by strictly following well-defined encoding rules described in Annex C. All attributes and composition/aggregation associations contained in the UML models are encoded either as XML elements or XML attributes but the names are fully consistent. One XML schema file is produced for each UML package.

Schematron patterns implement most additional requirements stated in Section 7. One Schematron file is produced for each XML package.

All example instances given in this section are informative and are used solely for illustrating features of the normative model. Many of these examples reference semantic information by using URLs that are resolvable to definitions maintained in one of several online ontologies:

- The general sensor community ontologies at http://sensorml.com/orr .
- The OGC online registry at http://www.opengis.net/def/.
- The SWEET ontology maintained by NASA JPL at http://sweet.jpl.nasa.gov/2.0/.
- The MMI ontology registry and repository at http://mmisw.org/orr/.

All XML examples are marked by a light gray background and formatted with a fixed-width font.

NOTE: These and additional examples, with explanation, can be found at:
http://www.sensorml.com/SensorML-2.0/examples/index.html

8.1 Requirements Class: Core Abstract Process Schema

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process	
Target Type	XML Instance
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process
Dependency	http://www.opengis.net/spec/SWE/2.0/req/xml-encoding-principles
Dependency	ISO 19136 (GML)
Dependency	OGC 08-094r1 (SWE Common Data)
Dependency	ISO 19139 (GMD)
Dependency	ISO 19139 (GCO)

XML Schema elements and types defined in the "*core.xsd*" schema file implements all classes defined in the "Core Abstract Process" UML package.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/schema-valid
Req 45. The XML instance shall be valid with respect to the XML grammar defined in the "core.xsd", as well as satisfy all Schematron patterns defined in "core.sch".

8.1.1 General XML Principles

This section lists common requirements associated to the XML encoding rules used in the context of this standard. As mentioned above, the normative XML schemas in this standard have been generated by strictly following UML to XML encoding rules, such that the schemas are the exact image of the UML models. The same encoding principles will be used by all extensions of this standard.

8.1.1.1 XML Encoding Conventions

The rules used to encode the SensorML models into an XML Schema are similar to those used to derive GML application schemas and defined in ISO 19136. Most extensions to these rules were defined and implemented within the OGC SWE Common Data v2.0 standard and have been defined to allow:

- Use of "soft-typed" properties. These properties are encoded as XML elements with a generic element name but provide a "*name*" attribute for further disambiguation.

- Encoding of certain properties as XML attributes. This type of encoding adds to the "element-only" rules defined by ISO 19136. It is restricted to properties with a primitive type and indicated by a new tagged value in the UML model.
- Use of a new abstract base type. A custom base type called "*AbstractSWEType"* is used for all complex types.

Following ISO 19136 encoding rules, each UML class with a <<Type>> or <<DataType>> stereotype, or no stereotype at all, is implemented in the schema as a global XML complex type with a corresponding global XML element (called object element). Each of these elements has the same name as the UML class (i.e. always UpperCamelCase) and the name of the associated complex type is a concatenation of this name and the word "Type".
Each UML class attribute is implemented either as a global complex type and a corresponding local element (called property element), or as an XML attribute. Each property complex type is given a name composed of the UML attribute name (always lowerCamelCase) and the words "PropertyType". The element is defined locally within the complex type representing the class carrying the attribute and named exactly like the attribute in UML (i.e. no global elements are created for class attributes). Class associations are implemented similarly except they cannot be implemented as an XML attributes.

8.1.1.2 IDs and Linkable Properties

The schemas defined in this standard make extensive use of "*xlink*" features to support hypertext referencing in XML. This allows most property elements to reference content either internally or externally to the instance document, instead of including this content inline. This is supported by extensive use of the "*id*" attribute (of type xs:ID) on most object elements, and of the "*swe:AssociationAttributeGroup*" attribute group, on most property elements.
According to settings in the SensorML models, values for properties can be provided inline, by reference, or by either method.
In properties that support "*xlink*" attributes, one can usually choose to define that property value inline, as in:

```
<swe:field>
  <swe:Quantity id="TEMP" … />
</swe:field>
```

One can then reference an object within the same document by its ID:

```
  <swe:field xlink:href="#TEMP"/>
```

An object within an external document can also be referenced by including the full URI:

```
<swe:field xlink:href="http://www.my.com/fields.xml#TEMP"/>
```

Typically, "*xlink*" references will be specified as resolvable URLs. It is required that the property has either the "*xlink:href*" attribute set or contain an inline value, even though this cannot be enforced by XML schema.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/ref-or-inline-value-present
Req 46. A property element supporting the "swe:AssociationAttributeGroup" shall either contain the value inline or populate the "xlink:href" attribute with a valid reference, but shall not be empty.

As later specified, further requirements may be placed on the use of objects reference by xlink. This includes in some case, the use of both *xlink:href* to provide a resolvable link to the object's description as well as the use of an *xlink:title* to provide the uniqueID of the object. Particular properties ay also require the application of *xlink:role* or *xlink:arcrole* to provide the absolute and relative role of the object, respectively.

8.1.1.3 Extensibility Points

The SensorML schemas define extensibility points that can be used to insert ad-hoc XML content that is not specified by this standard. This is done via optional "*extension*" property of type "*xs:anyType*" in the base abstract complex type "*AbstractSWEType*". Since all object types defined in this standard derive from this base type, extensions can be added for many properties in a SensorML instance.

This mechanism allows for a "laxist" way of including extended content in XML instances as the extended content is by default ignored by the validator. However, it is also possible to formally validate extended content by writing an XML schema for the extension and feeding it to the validator via the "*xsi:schemaLocation*" attribute in all instances using the extension.

The recommended way of extending the XML schema of this standard is to define new properties on existing objects by inserting them in an "*extension*" slot. Additionally this should be done in a way that these new properties can be safely ignored by an implementation that is not compatible with a given extension. However, it should be recognized that defining new XML object elements (such as new data component objects) rather than new properties will greatly reduce forward compatibility of implementations compliant to this standard with XML instances containing extensions of this standard.

In any case, all such *extensions* of the XML schema described in this standard will be defined in a new namespace (other than the namespaces used by this standard and its dependencies) in order to allow easy detection of extensions by implementations.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/extension-namespace-unique
Req 47. The value of the *extension* property shall be a schema in which the root element is defined in a new unique namespace (other than the namespaces used by this standard and its dependencies).

Extensions are not allowed to change the meaning or behavior of elements and types defined by this standard in any way (in this case, new classes or properties will be defined). This guarantees that implementations, which may have no knowledge of an extension, can still properly use XML instances containing these extensions.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/extension-coherent-with-core
Req 48. The value of the *extension* property shall not redefine or change the meaning or behavior of XML elements and types defined in this standard.

The execution of the process should not depend on information contained within an extension. Data values required for process execution will be provided within the input, output, and parameter properties, and within the SensorML and SWE Common Data namespace elements (i.e. sml and swe, respectively), and cannot exclusively be contained in the extension classes.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/extension-process-execution
Req 49. The value of the *extension* property shall not exclusively contain information required for the successful execution of the process.

Note that extension points are also supported by the SWE Common Data specification such that community-specific XML elements can be added to any property value that uses a SWE Common data type element as its value. Common anticipated applications of property-specific extension points include security tagging of individual properties and community-specific quality control characterization of property values.

8.1.2 General XSD Dependencies and XML Heading

The header for all specified schema provides namespaces and dependencies. The header for core processes below specifices the namespaces and dependent schema for “sml”, “swe”, “gml”, “gco”, and “gmd”.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:gco="http://www.isotc211.org/2005/gco" xmlns:gmd="http://www.isotc211.org/2005/gmd"
xmlns:sml="http://www.opengis.net/sensorML/2.0" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:swe="http://www.opengis.net/swe/2.0" targetNamespace="http://www.opengis.net/sensorML/2.0"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <annotation>
        <documentation>
            The core elements of an abstract process from which all major elements of SensorML are derived.
        </documentation>
    </annotation>
    <import namespace="http://www.isotc211.org/2005/gmd"
        schemaLocation="http://schemas.opengis.net/iso/19139/20070417/gmd/gmd.xsd"/>
    <import namespace="http://www.isotc211.org/2005/gco"
        schemaLocation="http://schemas.opengis.net/iso/19139/20070417/gco/gco.xsd"/>
    <import namespace="http://www.opengis.net/gml/3.2"
        schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
    <import namespace="http://www.opengis.net/swe/2.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/2.0/swe.xsd"/>
    <import namespace="http://www.w3.org/XML/1998/namespace"
        schemaLocation="http://www.w3.org/2001/xml.xsd"/>
```

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/gml-dependency
Req 50. This standard shall utilize and depend upon the OGC GML 3.2 standard schema within the http://www.opengis.net/gml/3.2 namespace

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/swe-common-dependency
Req 51. This standard shall utilize and depend upon the OGC SWE Common Data 2.0 standard within the http://www.opengis.net/swe/2.0 namespace

Several base abstract types are defined in the SWE Common Data standard are used for derivation of elements within this SensorML specification. These are defined in the "swe" namespace and defined in:

http://schemas.opengis.net/sweCommon/2.0/*basic_types.xsd*

These basic types are used as base substitution groups for all global XML elements defined in this standard. Below are XML schema snippets for the "*AbstractSWE*", "*AbstractSWEIdentifiable*" and "*AbstractSWEValue*" elements and corresponding complex types:

```
<element name="AbstractSWE" type="swe:AbstractSWEType" abstract="true">
    <annotation>
        <documentation>
            Base substitution groups for all SWE Common objects other than value objects
```

```
            </documentation>
        </annotation>
    </element>

    <complexType name="AbstractSWEType">
        <sequence>
            <element name="extension" type="anyType" minOccurs="0" maxOccurs="unbounded">
                <annotation>
                    <documentation>Extension slot for future extensions to this standard.</documentation>
                </annotation>
            </element>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
    </complexType>
```

The "*AbstractSWE*" complex type is the base for all derived complex types defined in this standard. It defines a first extension mechanism as an optional "*extension*" element that allows for any extended element content (in a namespace other than the SWE Common Data Model namespace). It also has an optional "*id*" attribute allowing referencing the object that derives from it.

```
    <element name="AbstractSWEIdentifiable" type="swe:AbstractSWEIdentifiableType" abstract="true"
        substitutionGroup="swe:AbstractSWE">
        <annotation>
            <documentation>
                Base substitution groups for all SWE Common objects with identification metadata
            </documentation>
        </annotation>
    </element>

    <complexType name="AbstractSWEIdentifiableType">
        <complexContent>
            <extension base="swe:AbstractSWEType">
                <sequence>
                    <element name="identifier" type="anyURI" minOccurs="0">
                        <annotation>
                            <documentation>
                                Unique identifier of the data component. It can be used to globally identify a particular
                                component of the dataset, a process input/output or a universal constant
                            </documentation>
                        </annotation>
                    </element>
                    <element name="label" type="string" minOccurs="0">
                        <annotation>
                            <documentation>
                                Textual label for the data component . This is often used for displaying a human readable
                                name for a dataset field or a process input/output
                            </documentation>
                        </annotation>
                    </element>
                    <element name="description" type="string" minOccurs="0">
                        <annotation>
                            <documentation>
                                Textual description (i.e. human readable) of the data component usually used to clarify its
                                nature
                            </documentation>
                        </annotation>
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

The "*AbstractSWEIdentifiable*" complex type derives from "*AbstractSWE*" and adds three identification elements. These elements are to be used as described in the UML section of this standard.

Based on these dependencies, an example header for a typical SensorML instance is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:PhysicalSystem gml:id="DAVIS_PRO_VANTAGE2"
  xmlns:sml="http://www.opengis.net/sensorML/2.0"
  xmlns:swe="http://www.opengis.net/swe/2.0"
  xmlns:gml="http://www.opengis.net/gml/3.2"
  xmlns:gmd="http://www.isotc211.org/2005/gmd"
  xmlns:gco="http://www.isotc211.org/2005/gco"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xsi:schemaLocation="http://www.opengis.net/sensorML/2.0 http://schemas.opengis.net/sensorml/2.0/sensorml.xsd">
```

8.1.3 DescribedObject Properties

The *DescribedObject* element is an XML Schema implementation of the UML classes defined in clause 7.2.2. *DescribedObject* is the base class for all process objects defined in this standard. *DescribedObject* is derived from *gml:AbstractFeature*. In essence it provides a common set of metadata for a general feature. It is particularly suited for processes and functional devices, as later described in this specification.
The XML snippet for the *DescribedObject* element and its corresponding complex types is shown below:

```
<element name="DescribedObject" type="sml:DescribedObjectType" abstract="true"
     substitutionGroup="gml:AbstractFeature">
    <annotation>
        <documentation>
            A feature with generic metadata which further clarifies the object and supports object discovery.
        </documentation>
    </annotation>
</element>

<complexType name="DescribedObjectType" abstract="true">
        <complexContent>
            <extension base="gml:AbstractFeatureType">
                <sequence>
                    <element name="extension" type="anyType" minOccurs="0" maxOccurs="unbounded">
                        <annotation>
                            <documentation>
                                A property that allows one to extend a document using a schema in a different
                                namespace from the current schema.
                            </documentation>
                        </annotation>
                    </element>
                    <element name="keywords" type="sml:KeywordListPropertyType"
                        minOccurs="0" maxOccurs="unbounded">
                        <annotation>
                            <documentation>
                                Short keywords describing the context of this document to aid in discovery.
                            </documentation>
                        </annotation>
```

```
</element>
<element name="identification" type="sml:IdentifierListPropertyType"
    minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            Identifiers useful for discovery of the process (e.g. short name, mission id, wing id, serial
            number, etc.)
        </documentation>
    </annotation>
</element>
<element name="classification" type="sml:ClassifierListPropertyType"
    minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            Classifiers useful for discovery of the process (e.g. process type, sensor type, intended
            application, etc.)
        </documentation>
    </annotation>
</element>
<element name="validTime" minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            The time instance or time range during which this instance description is valid.
        </documentation>
    </annotation>
    <complexType>
        <sequence>
            <group ref="sml:TimeInstantOrPeriod"/>
        </sequence>
    </complexType>
</element>
<element name="securityConstraints" type="anyType" minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            Overall security tagging of process description; individual tagging of properties can be
            done using extension element.
        </documentation>
    </annotation>
</element>
<element name="legalConstraints" type="gmd:MD_LegalConstraints_PropertyType"
    minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            Legal constraints applied to this description (e.g. copyrights, legal use, etc.)
        </documentation>
    </annotation>
</element>
<element name="characteristics" minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            Useful properties of this process that do not further qualify the output values (e.g.
            component dimensions, battery life, operational limits, etc).
        </documentation>
    </annotation>
    <complexType>
        <complexContent>
            <extension base="sml:CharacteristicListPropertyType">
                <attribute name="name" type="NCName" use="required"/>
            </extension>
        </complexContent>
    </complexType>
</element>
<element name="capabilities" minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            Properties that further clarify or quantify the output of the process (e.g. dynamic range,
```

```
                        sensitivity, threshold, etc.). These can assist in the discovery of processes that meet
                        particular requirements.
                    </documentation>
                </annotation>
                <complexType>
                    <complexContent>
                        <extension base="sml:CapabilityListPropertyType">
                            <attribute name="name" type="NCName" use="required"/>
                        </extension>
                    </complexContent>
                </complexType>
            </element>
            <element name="contacts" type="sml:ContactListPropertyType"
                minOccurs="0" maxOccurs="unbounded">
                <annotation>
                    <documentation>
                        Persons or responsible parties that are relevant to this process (e.g. designer,
                        manufacturer, expert, etc.)
                    </documentation>
                </annotation>
            </element>
            <element name="documentation" type="sml:DocumentListPropertyType"
                minOccurs="0" maxOccurs="unbounded">
                <annotation>
                    <documentation>
                        Additional external online documentation of relevance to this process (e.g. user's guides,
                        product manuals, specification sheets, images, technical papers, etc.)
                    </documentation>
                </annotation>
            </element>
            <element name="history" type="sml:EventListPropertyType"
                minOccurs="0" maxOccurs="unbounded">
                <annotation>
                    <documentation>
                        A collection of time-tagged events relevant to this process.
                    </documentation>
                </annotation>
            </element>
        </sequence>
        <attribute ref="xml:lang" use="optional">
            <annotation>
                <documentation>
                    A tag that identifies the language (e.g. english, french, etc.) for the overall document using a
                    two-letters code as defined by ISO 639-1.
                </documentation>
            </annotation>
        </attribute>
    </extension>
  </complexContent>
</complexType>
```

Most properties in *DescribedObject* are optional, thereby allowing the description to be as small or as robust as one desires.

All property elements in *DescribedObject* are of type *DescribedObjectPropertyType* which supports the ability to reference the value of the property externally through *xlink:href* or provide the value inline.

```
<complexType name="DescribedObjectPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:DescribedObject"/>
    </sequence>
```

```
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
        <attributeGroup ref="gml:OwnershipAttributeGroup"/>
    </complexType>
```

Additionally, most property values in *DescribedObject* are of type *AbstractMetadataListType,* which supports a list structure along with identification and definition properties.

```
<element name="AbstractMetadataList" type="sml:AbstractMetadataListType"
    substitutionGroup="swe:AbstractSWEIdentifiable"/>

<complexType name="AbstractMetadataListType">
    <complexContent>
        <extension base="swe:AbstractSWEIdentifiableType">
            <attribute name="definition" type="anyURI" use="optional"/>
        </extension>
    </complexContent>
</complexType>
<complexType name="AbstractMetadataListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:AbstractMetadataList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

8.1.3.1 Description Property

The *description* property is inherited from *gml:AbstractFeature* and provides a textual description for the feature.
An example of a description is given below:

```
<gml:description>Thermometer on the window of the Cass Building, Room 315</gml:description>
```

8.1.3.2 Name Property

The name property is inherited from *gml:AbstractFeature* and provides a common name for the feature.
An example of the name property is given below:

```
<gml:name>Health Physics Instruments 2070 Gamma Detector</gml:name>
```

8.1.3.3 Unique Identifier

This specification reserves the *gml:identifier*, inherited from *gml:AbstractFeature*, as a means of providing a unique identifier for the *DescribedObject*. This unique identifier should be referenced in any other specification that involves this object, thereby providing a means of searching for all references to this object.

Requirement

http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/globally-unique-id
Req 52. An instance supporting *DescribedObject* shall have one and only one *gml:identifier* and this element shall have a *codespace* attribute with a value set to "uniqueID". The value of the *gml:identifier* shall be a globally unique identifier for the *DescribedObject*.

The value of the gml:identifier can be a URI (e.g. URL or URN) or a series of string characters that uniquely identify the object. An example of a globally unique identifier for a weather station is given below:

```
<gml:identifier codeSpace="uid">urn:icd:stations:FR8766</gml:identifier>
```

```
<gml:identifier codeSpace="uid">38a7s8f9d55</gml:identifier>
```

8.1.3.4 Keywords

The *keyword* property and the *KeywordList* element are XML Schema implementations of the UML classes defined in clause 7.2.2.2.
The XML snippet for the *KeywordList* element and its corresponding complex types is shown below:

```
<element name="KeywordList" type="sml:KeywordListType" substitutionGroup="sml:AbstractMetadataList"/>

<complexType name="KeywordListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="codeSpace" type="swe:Reference" minOccurs="0" maxOccurs="1"/>
                <element name="keyword" type="string" minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="KeywordListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:KeywordList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

The *KeywordList* element includes an optional *codespace* attribute that should reference an external dictionary or keyword list that includes all possible keyword entries. An example of a keyword list without a codespace defined is shown below:

```
<keywords>
    <KeywordList>
        <keyword>weather station</keyword>
        <keyword>precipitation</keyword>
```

```
            <keyword>wind speed</keyword>
            <keyword>temperature</keyword>
        </KeywordList>
    </keywords>
```

An example of a keyword list with codespace is shown below:

```
<keywords>
    <KeywordList codespace="http://myAuthoritativeDomain.org/def/myKeywordList">
        <keyword>weather station</keyword>
        <keyword>precipitation</keyword>
        <keyword>wind speed</keyword>
        <keyword>temperature</keyword>
    </KeywordList>
</keywords>
```

8.1.3.5 Identifiers

The *IdentifierList* and the *Term* elements are XML Schema implementations of the UML classes defined in clause 7.2.2.3. The *identifier* property of the *IdentifierList* takes a *Term* as its value. The *Term* element has an optional *definition* attribute that should reference a resolvable definition of the term within an online dictionary or ontology.
The XML snippets for the *IdentifierList* and *Term* elements and their corresponding complex types are shown below:

Term:

```
<element name="Term" type="sml:TermType" substitutionGroup="swe:AbstractSWE"/>

<complexType name="TermType">
    <complexContent>
        <extension base="swe:AbstractSWEType">
            <sequence>
                <element name="label" type="string"/>
                <element name="codeSpace" type="swe:Reference" minOccurs="0" maxOccurs="1"/>
                <element name="value" type="string"/>
            </sequence>
            <attribute name="definition" type="anyURI" use="optional"/>
        </extension>
    </complexContent>
</complexType>
<complexType name="TermPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:Term"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

IdentifierList:

```
<element name="IdentifierList" type="sml:IdentifierListType" substitutionGroup="sml:AbstractMetadataList"/>

<complexType name="IdentifierListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
```

```
                    <element name="identifier" minOccurs="1" maxOccurs="unbounded">
                        <complexType>
                            <sequence>
                                <element ref="sml:Term"/>
                            </sequence>
                        </complexType>
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>

    <complexType name="IdentifierListPropertyType">
        <sequence minOccurs="0">
            <element ref="sml:IdentifierList"/>
        </sequence>
        <attributeGroup ref="swe:AssociationAttributeGroup"/>
    </complexType>
```

An example of an identifier list is shown below:

```
<sml:identification>
    <sml:IdentifierList>
      <sml:identifier>
        <sml:Term definition="http://sensorml.com/ont/swe/property/ShortName">
          <sml:label>Short Name</sml:label>
          <sml:value>Thermometer FR8766</sml:value>
        </sml:Term>
      </sml:identifier>
      <sml:identifier>
        <sml:Term definition="http://sensorml.com/ont/swe/property/Manufacturer">
          <sml:label>Manufacturer Name</sml:label>
          <sml:value>ACME Inc</sml:value>
        </sml:Term>
      </sml:identifier>
      <sml:identifier>
        <sml:Term definition="http://sensorml.com/ont/swe/property/ModelNumber">
          <sml:label>Manufacturer Model</sml:label>
          <sml:value>T911</sml:value>
        </sml:Term>
      </sml:identifier>
      <sml:identifier>
        <sml:Term definition=" http://sensorml.com/ont/swe/property/SerialNumber">
          <sml:label>Serial Number</sml:label>
          <sml:value>FT5743456566-997</sml:value>
        </sml:Term>
      </sml:identifier>
    </sml:IdentifierList>
  </sml:identification>
```

8.1.3.6 Classifiers

The *ClassifierList* is an XML Schema implementations of the UML classes defined in clause 7.2.2.4. The *classifier* property of the *ClassifierList* takes a *Term* as its value.
The XML snippet for the *ClassifierList* element and its corresponding complex types is shown below:

```
    <element name="ClassifierList" type="sml:ClassifierListType" substitutionGroup="sml:AbstractMetadataList"/>
```

```
<complexType name="ClassifierListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="classifier" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <sequence>
                            <element ref="sml:Term"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ClassifierListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ClassifierList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

An example of a classifier list is shown below:

```
<sml:classification>
  <sml:ClassifierList>
    <sml:classifier>
      <sml:Term definition="http://sensorml.com/ont/swe/property/SensorType">
        <sml:label>Sensor Type</sml:label>
        <sml:value>Weather Station</sml:value>
      </sml:Term>
    </sml:classifier>
      <sml:classifier>
      <sml:Term definition="http://sensorml.com/ont/swe/property/IntendedApplication">
        <sml:label>Intended Application</sml:label>
        <sml:value>Weather</sml:value>
      </sml:Term>
    </sml:classifier>
  </sml:ClassifierList>
</sml:classification>
```

8.1.3.7 Security Constraints

The *securityConstraints* property provides an overall security tagging for the overall document. Typically if any part of a document is classified as Secret, for instance, then the entire document is tagged as Secret.

Because various nations and other entities may have developed their own XML schema for supporting security tagging, the value of the *securityConstraints* property is provide as *xs:any*. Thus, like the *extension* property, the value of the *securityConstraint* property can be any XML, but this XML will be defined in a namespace other than those used in this standard.

Requirement

http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/document-security-tags
Req 53. All values for the *securityConstraints* property shall be defined in a new unique namespace (other than the namespaces used by this standard and its dependencies).

8.1.3.8 Note on Security Tagging Individual Properties

It is often required or desireable that security tagging exists not only for the entire document but for individual property values as well. SensorML supports tagging of individual property values and lists through the *extension* property that is inherited by all elements derived from *AbstractSWEType* and *AbstractSWEIdentifiableType*.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/individual-security-tags
Req 54. Security tagging of individual property values shall utilize the *extension* element within the appropriate SensorML or SWE Common data elements.

8.1.3.9 Valid Time Constraints

The *validTime* element is an XML Schema implementation of the UML class defined in clause 7.2.2.6.

The XML Schema snippet from *DescribedObject* that pertains to the *validTime* property is given below. The *validTime* property takes either a GML *TimePeriod* or *TimeInstant* as its value.

```
<element name="validTime" minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>
            The time instance or time range during which this instance description is valid.
        </documentation>
    </annotation>
    <complexType>
        <sequence>
            <group ref="sml:TimeInstantOrPeriod"/>
        </sequence>
    </complexType>
</element>

<group name="TimeInstantOrPeriod">
    <annotation>
        <documentation>Either a Time Instant or Time Period</documentation>
    </annotation>
    <choice>
        <element ref="gml:TimePeriod"/>
        <element ref="gml:TimeInstant"/>
    </choice>
</group>

<complexType name="TimeInstantOrPeriodPropertyType">
    <sequence minOccurs="0">
        <group ref="sml:TimeInstantOrPeriod"/>
```

```
        </sequence>
        <attributeGroup ref="swe:AssociationAttributeGroup"/>
    </complexType>
```

An example of the valid time being given by a TimePeriod is shown below:

```
<validTime>
    <gml:TimePeriod gml:id="deploymentDates">
        <gml:beginPosition>2009-01-01T14:00:00Z</gml:beginPosition>
        <gml:endPosition>2013-12-31T08:20:00Z</gml:endPosition>
    </gml:TimePeriod>
</validTime>
```

8.1.3.10 Legal Constraints

The *legalConstraints* element is an XML Schema implementation of the UML class defined in clause 7.2.2.7.

The XML Schema snippet from *DescribedObject* that pertains to the *legalConstraints* property is given below. The *legalConstraints* property takes an ISO 19115 *MD_LegalConstraints* element as its value, which is encoded according to ISO19139 schema.

```
<element name="legalConstraints" type="gmd:MD_LegalConstraints_PropertyType"
    minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <documentation>Legal constraints applied to this description (e.g. copyrights, legal use, etc.)
        </documentation>
    </annotation>
</element>
```

An example of legalConstraints is provided below:

```
<sml:legalConstraints>
    <gmd:MD_LegalConstraints>
        <gmd:useLimitation>
            <gco:CharacterString>
            Disclaimer - While every effort has been made to ensure that the data from this sensor
            is accurate and reliable within the limits of the current state of the art, we cannot assume
            liability for any damages caused by any errors or omissions in the data, nor as a result of
            the failure of the data to function on a particular system. We makes no warranty, expressed
            or implied, nor does the fact of distribution constitute such a warranty.
            </gco:CharacterString>
        </gmd:useLimitation>
    </gmd:MD_LegalConstraints>
</sml:legalConstraints>
```

8.1.3.11 Capabilities

The *capabilities* property and the *CapabilitiesList* element are XML Schema implementations of the UML classes defined in clause 7.2.2.9. The *capabilities* property

takes a *CapabilitiesList* as its value. The *CapabilitiesList member* property accepts any SWE Common data component as its value.
The XML snippet for the *CapabilitiesList* element and its corresponding complex types is shown below:

```
<element name="CapabilityList" type="sml:CapabilityListType" substitutionGroup="sml:AbstractMetadataList"/>

<complexType name="CapabilityListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="capability" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <complexContent>
                            <extension base="swe:AbstractDataComponentPropertyType">
                                <attribute name="name" type="NCName" use="required"/>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="CapabilityListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:CapabilityList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

Thus capabilities can be described using SWE Common simple types, such as *Quantity*, *Count*, *Boolean*, *Category*, *Text*, and *Time* as well as be aggregate types such as *DataRecord*, *DataArray*, *Vector*, or *Matrix*.
An example of *capabilities* is given below:

```
<sml:capabilities>
  <sml:CapabilityList>
    <sml:capability name="measurementProperties">
      <swe:DataRecord definition="http://sensorml.com/ont/swe/property/MeasurementProperties">
        <swe:label>Measurement Properties</swe:label>
        <swe:field name="RadiationRange">
          <swe:QuantityRange definition="http://sensorml.com/ont/swe/property/RadiationLevel">
            <swe:uom code="R/h"/>
            <swe:value>0 30</swe:value>
          </swe:QuantityRange>
        </swe:field>
        <swe:field name="Sensitivitity">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/RadiationSensitivity">
            <swe:uom code="{tot}/uR"/>
            <swe:value>1</swe:value>
          </swe:Quantity>
        </swe:field>
        <swe:field name="SamplePeriod">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/SamplePeriod">
            <swe:uom code="s"/>
            <swe:value>1</swe:value>
          </swe:Quantity>
        </swe:field>
        <swe:field name="MeasurementOutputTime">
```

```
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/OutputPeriod">
              <swe:uom code="s"/>
              <swe:value>450</swe:value>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </sml:capability>
    </sml:CapabilityList>
</sml:capabilities>
```

8.1.3.12 Characteristics

The *characteristics* property and the *CharacteristicsList* element are XML Schema implementations of the UML classes defined in clause 7.2.2.9. The *characteristics* property takes a *CharacteristicsList* as its value. The *CharacteristicsList member* property accepts any SWE Common data component as its value.

The XML snippet for the *CharacteristicsList* element and its corresponding complex types is shown below:

```
<element name="CharacteristicList" type="sml:CharacteristicListType"
    substitutionGroup="sml:AbstractMetadataList"/>

<complexType name="CharacteristicListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="characteristic" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <complexContent>
                            <extension base="swe:AbstractDataComponentPropertyType">
                                <attribute name="name" type="NCName" use="required"/>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="CharacteristicListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:CharacteristicList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

Thus characteristics can be described using SWE Common simple types, such as *Quantity*, *Count*, *Boolean*, *Category*, *Text*, and *Time* as well as be aggregate types such as *DataRecord*, *DataArray*, *Vector*, or *Matrix*.

An example of *characteristics* that groups like properties in data records is given below:

```
<sml:characteristics name="generalProperties">
    <sml:CharacteristicList>

        <sml:characteristic name="physicalProperties">
            <swe:DataRecord definition="http://sensorml.com/ont/swe/property/PhysicalProperties">
                <swe:label>Physical Properties</swe:label>
```

```
                <swe:field name="PhysicalProperties">
                    <swe:DataRecord>
                        <swe:field name="Weight">
                            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Weight">
                                <swe:uom code="oz"/>
                                <swe:value>10</swe:value>
                            </swe:Quantity>
                        </swe:field>
                        <swe:field name="Length">
                            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Length">
                                <swe:uom code="in"/>
                                <swe:value>4.5</swe:value>
                            </swe:Quantity>
                        </swe:field>
                        <swe:field name="Width">
                            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Width">
                                <swe:uom code="in"/>
                                <swe:value>2.5</swe:value>
                            </swe:Quantity>
                        </swe:field>
                        <swe:field name="Height">
                            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Height">
                                <swe:uom code="in"/>
                                <swe:value>1.4</swe:value>
                            </swe:Quantity>
                        </swe:field>
                        <swe:field name="CasingMaterial">
                            <swe:Category definition="http://sensorml.com/ont/swe/property/Material">
                                <swe:value>Aluminum</swe:value>
                            </swe:Category>
                        </swe:field>
                    </swe:DataRecord>
                </swe:field>
            </swe:DataRecord>
        </sml:characteristic>

        <sml:characteristic name="electricalRequirements">
            <swe:DataRecord definition="http://sensorml.com/ont/swe/property/PowerRequirement">
                <swe:label>Electrical Requirements</swe:label>
                <swe:field name="voltage">
                    <swe:QuantityRange definition="http://sensorml.com/ont/swe/property/Voltage">
                        <swe:uom code="V"/>
                        <swe:value>8 12</swe:value>
                    </swe:QuantityRange>
                </swe:field>
                <swe:field name="CurrentType">
                    <swe:Category definition="http://sensorml.com/ont/swe/property/ElectricalCurrentType">
                        <swe:value>DC</swe:value>
                    </swe:Category>
                </swe:field>
                <swe:field name="AmpRange">
                    <swe:QuantityRange definition="http://sensorml.com/ont/swe/property/ElectricalCurrent">
                        <swe:uom code="mA"/>
                        <swe:value>20 40</swe:value>
                    </swe:QuantityRange>
                </swe:field>
            </swe:DataRecord>
        </sml:characteristic>

    </sml:CharacteristicList>
</sml:characteristics>
```

8.1.3.13 Contacts

The *contacts* property and the *ContactList* element are XML Schema implementations of the UML classes defined in clause 7.2.2.11. The *contacts* property takes a *ContactList* as its value. The *ContactList member* property supports an ISO 19115 *CI_ResponsibleParty* as its value, which is encoded according to ISO19139 schema.
The XML snippet for the *ContactList* element and its corresponding complex types is shown below:

```
<element name="ContactList" type="sml:ContactListType" substitutionGroup="sml:AbstractMetadataList"/>

<complexType name="ContactListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="contact" type="gmd:CI_ResponsibleParty_PropertyType"
                    minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ContactListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ContactList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

A contact can have multiple roles for which it is responsible. What is of most interest in this standard is the role the responsible party plays relative to the object being described. Therefore, the role of the responsible party (e.g. manufacturer, expert, owner, etc.) should be given by the *xlink:arcrole* attribute in the *member* property of the *ContactList*.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/contact-role
Req 55. The role of the responsible party relative to the described object shall be provided using the *xlink:arcrole* attribute of the relevant *member* property of the *ContactList*.

An example of a *ContactList* is shown below (note that the *gmd* schema requires a *CharacterString* element before the values of properties).

```
<sml:contacts>
  <sml:ContactList>

    <sml:contact xlink:arcrole="http://sensorml.com/ont/swe/role/Operator">
      <gmd:CI_ResponsibleParty>
        <gmd:organisationName>
          <gco:CharacterString>METEO France</gco:CharacterString>
        </gmd:organisationName>
        <gmd:contactInfo>
          <gmd:CI_Contact>
            <gmd:phone>
```

```
            <gmd:CI_Telephone>
              <gmd:voice>
                <gco:CharacterString>+33 5 99 11 22 33 44</gco:CharacterString>
              </gmd:voice>
            </gmd:CI_Telephone>
          </gmd:phone>
          <gmd:address>
            <gmd:CI_Address>
              <gmd:deliveryPoint>
                <gco:CharacterString>42 Avenue Gaspard Coriolis</gco:CharacterString>
              </gmd:deliveryPoint>
              <gmd:city>
                <gco:CharacterString>TOULOUSE</gco:CharacterString>
              </gmd:city>
              <gmd:postalCode>
                <gco:CharacterString>31100</gco:CharacterString>
              </gmd:postalCode>
              <gmd:country>
                <gco:CharacterString>FRANCE</gco:CharacterString>
              </gmd:country>
            </gmd:CI_Address>
          </gmd:address>
        </gmd:CI_Contact>
      </gmd:contactInfo>
      <gmd:role gco:nilReason="inapplicable"/>
    </gmd:CI_ResponsibleParty>
  </sml:contact>

  <sml:contact
      xlink:arcrole="http://sensorml.com/ont/swe/role/Manufacturer"
      xlink:href="http://www.myCompany.com/contact/company.xml"/>

 </sml:ContactList>
</sml:contacts>
```

8.1.3.14 Documentation

The *documentation* property and the *DocumentList* element are XML Schema implementations of the UML classes defined in clause 7.2.2.11. The *documentation* property takes a *DocumentList* as its value. The *DocumentList member* property supports an ISO 19115 *CI_OnlineResource* as its value, which is encoded according to ISO19139 schema.

The XML snippet for the *DocumentList* element and its corresponding complex types is shown below:

```
<element name="DocumentList" type="sml:DocumentListType" substitutionGroup="sml:AbstractMetadataList"/>

<complexType name="DocumentListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="document" type="gmd:CI_OnlineResource_PropertyType"
                    minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="DocumentListPropertyType">
```

```
        <sequence minOccurs="0">
            <element ref="sml:DocumentList"/>
        </sequence>
        <attributeGroup ref="swe:AssociationAttributeGroup"/>
    </complexType>
```

An example of *documentation* is given below (note that the *gco* and *gmd* schema require a *CharacterString* element before the values of properties):

```
    <documentation>
        <DocumentList>
            <document xlink:arcrole="http://sensorml.com/ont/swe/role/UserManual">
                <gmd:CI_OnlineResource>
                    <gmd:linkage>
                        <gmd:URL>http://myCompany.com/ref/2031manual.pdf</gmd:URL>
                    </gmd:linkage>
                    <gmd:name>
                        <gco:CharacterString>
                            User Manual for Model 2031
                        </gco:CharacterString>
                    </gmd:name>
                    <gmd:description>
                        <gco:CharacterString>
                            This document provides the complete Users Manual for the myCompany sensor model 2031.
                        </gco:CharacterString>
                    </gmd:description>
                </gmd:CI_OnlineResource>
            </document>

            <document xlink:arcrole="http://sensorml.com/ont/swe/role/ProductImage">
                <gmd:CI_OnlineResource>
                    <gmd:linkage>
                        <gmd:URL>http://myCompany.com/ref/2031image.jpg</gmd:URL>
                    </gmd:linkage>
                    <gmd:name>
                        <gco:CharacterString>
                            Sensor Model 2031
                        </gco:CharacterString>
                    </gmd:name>
                    <gmd:description>
                        <gco:CharacterString>
                            This is an image of sensor model 2031.
                        </gco:CharacterString>
                    </gmd:description>
                </gmd:CI_OnlineResource>
            </document>

        </DocumentList>
    </documentation>
```

8.1.3.15 History

The *history* property and the *EventList* and *Event* elements are XML Schema implementations of the UML classes defined in clause 7.2.2.12. The *history* property takes an *EventList* as its value. The *EventList member* property takes an *Event* as its value. The XML snippets for the *EventList* and *Event* elements and their corresponding complex types are shown below:

EventList:

```
    <element name="EventList" type="sml:EventListType" substitutionGroup="sml:AbstractMetadataList"/>
```

```
<complexType name="EventListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="event" type="sml:EventPropertyType"
                    minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="EventListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:EventList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

Event:

```
<element name="Event" type="sml:EventType" substitutionGroup="swe:AbstractSWEIdentifiable">
    <annotation>
        <documentation>A time tagged Event with description and relevant property values.</documentation>
    </annotation>
</element>
<complexType name="EventType">
    <complexContent>
        <extension base="swe:AbstractSWEIdentifiableType">
            <sequence>
                <element name="keywords" type="sml:KeywordListPropertyType" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>keywords useful for discovery of the event</documentation>
                    </annotation>
                </element>
                <element name="identification" type="sml:IdentifierListPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>Identifiers relevant to the event</documentation>
                    </annotation>
                </element>
                <element name="classification" type="sml:ClassifierListPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>Type of event (useful for discovery)</documentation>
                    </annotation>
                </element>
                <element name="contacts" type="sml:ContactListPropertyType" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>Persons or parties relevant to this event</documentation>
                    </annotation>
                </element>
                <element name="documentation" type="sml:DocumentListPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>Additional documentation relevant to this event</documentation>
                    </annotation>
                </element>
                <element name="time">
                    <annotation>
                        <documentation>DateTime of the event</documentation>
                    </annotation>
                    <complexType>
                        <sequence>
                            <group ref="sml:TimeInstantOrPeriod"/>
                        </sequence>
                    </complexType>
```

```
                </element>
                <element name="property" type="swe:AbstractDataComponentPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            Properties of interest to the event (e.g. calibration values, condition category, error codes,
                            etc)
                        </documentation>
                    </annotation>
                </element>
                <element name="configuration" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>Configuration settings adjusted during event</documentation>
                    </annotation>
                    <complexType>
                        <sequence>
                            <element ref="sml:AbstractSettings"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
            <attribute name="definition" type="anyURI" use="optional"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="EventPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:Event"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

An example of an *EventList* is shown below. *EventList* could of course be presented inline as a value of the *history* property, or could be maintained externally and referenced by the *xlink:href* attribute of the *history* property.

```
<sml:EventList>
 <sml:event>
  <sml:Event>
   <swe:label>Scheduled Maintenance</swe:label>
   <swe:description>
       Monthly maintenance of station exterior.&#13;-Checked electronics&#13;-Checked casing&#13;Checked power
       supply.&#13;Everything OK.
    </swe:description>
   <sml:time>
    <gml:TimePeriod gml:id="MP1">
     <gml:beginPosition>2002-03-01T10:00:00Z</gml:beginPosition>
     <gml:endPosition>2002-03-01T11:00:00Z</gml:endPosition>
    </gml:TimePeriod>
   </sml:time>
  </sml:Event>
 </sml:event>
 <sml:event>
  <sml:Event>
   <swe:label>Calibration</swe:label>
   <swe:description>Recalibration of acquisition electronics using temperature reference</swe:description>
   <sml:time>
    <gml:TimeInstant gml:id="MP2">
     <gml:timePosition>2002-03-01T18:00:00Z</gml:timePosition>
    </gml:TimeInstant>
   </sml:time>
   <sml:configuration>
```

```
      <sml:Settings>
        <sml:setArrayValues ref="base/components/raingauge/parameters/steady-state-response">
           1,2,3,4,5 2,4,6,8,10
        </sml:setArrayValues>
      </sml:Settings>
    </sml:configuration>
  </sml:Event>
 </sml:event>
</sml:EventList>
```

8.1.4 Abstract Process

AbstractProcessType is derived from *DescribedObject* and serves as the base class for all processes modelled and encoded in this specification. Thus, all processes include the metadata described above plus the elements defined in this section and its subsections.

The *AbstractProcess* element is an XML Schema implementation of the UML class defined in clause 7.2.3.

The XML snippet for the *AbstractProcess* element and its corresponding complex types is shown below. The various properties of *AbstractProcess* will be discussed in more detail in the following subsections.

```
<element name="AbstractProcess" type="sml:AbstractProcessType" abstract="true"
    substitutionGroup="sml:DescribedObject">
    <annotation>
        <documentation>The general base model for any process.</documentation>
    </annotation>
</element>
<complexType name="AbstractProcessType" abstract="true">
    <complexContent>
        <extension base="sml:DescribedObjectType">
            <sequence>
                <element name="typeOf" type="gml:ReferenceType" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <appinfo>
                            <gml:targetElement>sml:AbstractProcess</gml:targetElement>
                        </appinfo>
                        <documentation>
                            A reference to a base process from which this process inherits properties and constraints
                            (e.g. original equipment manufacturer's model description, generic equation, etc.). The
                            uniqueID of the referenced process must be provided using the xlink:title attribute while
                            the URL to the process description must be provided by the xlink:href attribute.
                        </documentation>
                    </annotation>
                </element>
                <element name="configuration" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            Value settings that further constrain the properties of the base process.
                        </documentation>
                    </annotation>
                    <complexType>
                        <sequence>
                            <element ref="sml:AbstractSettings"/>
                        </sequence>
                    </complexType>
                </element>
                <element name="featuresOfInterest" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            A collection of features relevant to a process (e.g. the Gulf of Mexico, the White House,
```

```
                    the set of all Fibonacci Numbers, etc.); can also support a sampling feature. The primary
                    purpose of the Features of Interest is to support discovery.
                </documentation>
            </annotation>
            <complexType>
                <sequence>
                    <element ref="sml:FeatureList"/>
                </sequence>
            </complexType>
        </element>
        <element name="inputs" minOccurs="0" maxOccurs="1">
            <annotation>
                <documentation>
                    The list of data components (and their properties and semantics) that the process will
                    accept as input; In the standard linear equation y=mx+b; x is the input, m and b are the
                    parameters, and y is the output.
                </documentation>
            </annotation>
            <complexType>
                <sequence>
                    <element ref="sml:InputList"/>
                </sequence>
            </complexType>
        </element>
        <element name="outputs" minOccurs="0" maxOccurs="1">
            <annotation>
                <documentation>
                    The list of data components (and their properties and semantics) that the process will
                    accept as output; In the standard linear equation y=mx+b; x is the input, m and b are the
                    parameters, and y is the output.
                </documentation>
            </annotation>
            <complexType>
                <sequence>
                    <element ref="sml:OutputList"/>
                </sequence>
            </complexType>
        </element>
        <element name="parameters" minOccurs="0" maxOccurs="1">
            <annotation>
                <documentation>
                    The list of data components (and their properties and semantics) that the process will
                    accept as parameters; In the standard linear equation y=mx+b; x is the input, m and b are
                    the parameters, and y is the output.
                </documentation>
            </annotation>
            <complexType>
                <sequence>
                    <element ref="sml:ParameterList"/>
                </sequence>
            </complexType>
        </element>
        <element name="modes" minOccurs="0" maxOccurs="unbounded">
            <annotation>
                <documentation>
                    A collection of parameters that can be set at once through the selection of a particular
                    predefined mode.
                </documentation>
            </annotation>
            <complexType>
                <sequence>
                    <element ref="sml:AbstractModes"/>
                </sequence>
            </complexType>
        </element>
    </sequence>
```

```
            <attribute name="definition" type="anyURI">
                <annotation>
                    <documentation>
                        An optional property that allows one to reference the process instance in an online ontology
                        or dictionary. The value of the property must be a resolvable URI.
                    </documentation>
                </annotation>
            </attribute>
        </extension>
    </complexContent>
</complexType>

<complexType name="AbstractProcessPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:AbstractProcess"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

8.1.4.1 Definition

As with the *definition* attribute used in SWE Common simple and aggregate data components, the *definition* attribute for a *DescribedObject* provides the ability to link to a definition or classification within an online ontology. Relationships of this object with various other objects can then be inferred based on the ontology. There is some potential for overlap of functionality between the *definition* attribute and the *identification* and *classification* properties. Some communities may prefer to primarily use *classification* and *identification* elements, while others prefer to use the *definition* attribute or both.

8.1.4.2 TypeOf

The *typeOf* property provides a reference to a base process from which this process inherits properties and constraints. It is a key component in support of inheritance as described in the models in Clause 7.2.3.3.

Two pieces of information required in order to reference a base process instance, the uniqueID of the process and a resolvable URL reference to the process description. These should be provided by the *xlink:title* and *xlink:href* attributes, respectively.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/type-of-reference
Req 56. The *typeOf* property shall require meaningful values for the *xlink:title* and *xlink:href* attributes. The uniqueID of the referenced process, given by its *gml:identifier* property, shall be the value of the *xlink:title* attribute, while a resolvable URL to the reference process description shall be the value of the *xlink:href* attribute.

An example of the *typeOf* property is given below:

```
<sml:typeOf xlink:title="urn:heath:2070"
```

```
xlink:href="http://www.sensors.ws/examples/sensorml-2.0/configuration/gamma2070.xml"/>
```

8.1.4.3 Abstract Configuration

The optional *configuration* property takes an *AbstractConfiguration* as its value, which provides a placeholder for configuration information in the *AbstractProcess*. A concrete class supporting configuration will be provided in a higher-level conformance class in Requirements Class: Configurable Process Schema in Clause 8.6.

8.1.4.4 FeatureOfInterest

The *featureOfInterest* property and the *FeatureList* element are XML Schema implementations of the UML classes defined in clause 7.2.3.1. The *featureOfInterest* property takes a *FeatureList* as its value. The *FeatureList member* property takes any object derived from GML *AbstractFeature* as its value. This feature can be described inline, but will most often be provided by reference using the *xlink:href* attribute.
The XML snippet for the *FeatureList* element and its corresponding complex types is shown below.

```
<element name="FeatureList" type="sml:FeatureListType" substitutionGroup="sml:AbstractMetadataList"/>

<complexType name="FeatureListType">
    <complexContent>
        <extension base="sml:AbstractMetadataListType">
            <sequence>
                <element name="feature" type="gml:FeaturePropertyType"
                    minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="FeatureListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:FeatureList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

An example for a Feature-of-Interest list is given below. In this case, all features are by reference.

```
<sml:featureOfInterest>
      <sml:FeatureList>
          <sml:feature xlink:arcrole="http://sensorml.com/ont/swe/feature/Station"
              xlink:title="urn:myDomain:station:A209"
              xlink:href="http://myDomain.com/features/officeA209.xml"/>
          <sml:feature xlink:arcrole="http://sensorml.com/ont/swe/feature/ObservedFeature"
              xlink:title="urn:mmi:features:GulfOfMexico"
              xlink:href="http://mmi.org/features/GulfOfMexico.xml"/>
      </sml:FeatureList>
  </sml:featureOfInterest>
```

To better understand the role that the feature of interest plays with regard to the process, the *xlink:arcrole* attribute of the *feature* property should be used to specify this relationship. To aid in discovery, the unique ID or name of the feature should be provided in the *xlink:title* attribute if it exists.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/foi-arcrole-and-title
Req 57. Regardless of whether a feature of interest is described inline or provided by reference, the *xlink:arcrole* attribute of the *member* property of the *FeatureList* shall be used to specify the relationship of the associated feature to the process. If a unique identifier or name for the feature of interest exists, it shall be the value of the *xlink:title* of the *member* property of the *FeatureList*.

8.1.4.5 Inputs, Outputs, and Parameters

The *input, output,* and *parameter* properties and the *InputList, OutputList,* and *ParameterList* elements are XML Schema implementations of the UML classes defined in clause 7.2.3.1. The *input, output,* and *parameter* properties takes *InputList, OutputList,* and *ParameterList* elements as their respective value. The *member* properties of these lists take any element of the *DataComponentOrObservable* Union as their value. This union and its components will be discussed in more detail in subsequent sections.
The XML snippets for the *InputList, OutputList,* and *ParameterList* elements and their corresponding complex types are shown below:

Inputs:

```
<element name="InputList" type="sml:InputListType" substitutionGroup="swe:AbstractSWE"/>

<complexType name="InputListType">
    <complexContent>
        <extension base="swe:AbstractSWEType">
            <sequence>
                <element name="input" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <complexContent>
                            <extension base="sml:DataComponentOrObservablePropertyType">
                                <attribute name="name" type="NCName" use="required"/>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="InputListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:InputList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

Outputs:

```
<element name="OutputList" type="sml:OutputListType" substitutionGroup="swe:AbstractSWE"/>

<complexType name="OutputListType">
    <complexContent>
        <extension base="swe:AbstractSWEType">
            <sequence>
                <element name="output" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <complexContent>
                            <extension base="sml:DataComponentOrObservablePropertyType">
                                <attribute name="name" type="NCName" use="required"/>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="OutputListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:OutputList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

Parameters:

```
<element name="ParameterList" type="sml:ParameterListType" substitutionGroup="swe:AbstractSWE"/>

<complexType name="ParameterListType">
    <complexContent>
        <extension base="swe:AbstractSWEType">
            <sequence>
                <element name="parameter" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <complexContent>
                            <extension base="sml:DataComponentOrObservablePropertyType">
                                <attribute name="name" type="NCName" use="required"/>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ParameterListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ParameterList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

An example of inputs, outputs, and parameters from a particular process is given below (in this case, a gamma radiation sensor):

```
<!-- ================== -->
```

```
<!--   Inputs              -->
<!-- ==================== -->
<sml:inputs>
    <sml:InputList>
        <sml:input name="gammaRadiation">
            <sml:ObservableProperty definition="http://sensorml.com/ont/swe/property/Radiation">
                <swe:label>Electromagnetic Radiation</swe:label>
            </sml:ObservableProperty>
        </sml:input>
    </sml:InputList>
</sml:inputs>
<!-- ==================== -->
<!--    outputs              -->
<!-- ==================== -->
<sml:outputs>
    <sml:OutputList>
        <sml:output name="dose">
            <swe:DataRecord>
                <swe:field name="averageDose">
                    <swe:Quantity definition="http://sensorml.com/ont/swe/property/DoseRate.html">
                        <swe:label>Average Dose of Gamma Radiation</swe:label>
                        <swe:uom code="uR/min"/>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="InstantaneousDose">
                    <swe:Quantity definition="http://sensorml.com/ont/swe/property/RadiationDose">
                        <swe:label>Instantaneous Dose of Gamma Radiation</swe:label>
                        <swe:uom code="uR"/>
                    </swe:Quantity>
                </swe:field>
            </swe:DataRecord>
        </sml:output>
    </sml:OutputList>
</sml:outputs>
<!-- ======================== -->
<!--       Parameters             -->
<!-- ======================== -->
<sml:parameters>
    <sml:ParameterList>
        <sml:parameter name="energyResponse">
            <swe:DataArray definition="http://sensorml.com/ont/swe/property/SpectralResponse">
                <swe:label>Gamma Radiation Response Curve</swe:label>
                <swe:description>
                    The normalized energy response per KeV showing the sensitivity to gamma radiation
                </swe:description>
                <swe:elementCount>
                    <swe:Count>
                        <swe:value>7</swe:value>
                    </swe:Count>
                </swe:elementCount>
                <swe:elementType name="energyResponse">
                    <swe:DataRecord>
                        <swe:label>Normalize Energy Response</swe:label>
                        <swe:field name="radiationEnergy">
                            <swe:Quantity definition="http://sensorml.com/ont/swe/property/RadiationEnergy">
                                <swe:label>Radiation Energy</swe:label>
                                <swe:uom code="KeV"/>
                            </swe:Quantity>
                        </swe:field>
                        <swe:field name="relativeResponse">
                            <swe:Quantity definition="http://sensorml.com/ont/swe/property/RelativeResponse">
                                <swe:label>Relative Response</swe:label>
                                <swe:uom code="%"/>
                            </swe:Quantity>
                        </swe:field>
                    </swe:DataRecord>
```

```
                    </swe:elementType>
                    <swe:encoding>
                        <swe:TextEncoding blockSeparator="&#10;" tokenSeparator=","/>
                    </swe:encoding>
                    <swe:values>
                        10.0,73 17.5,89.5 20.0,94.0 30.,95.5 40.0,96.0 50.0,96.0 100.0,94.0
                    </swe:values>
                </swe:DataArray>
            </sml:parameter>
        </sml:ParameterList>
    </sml:parameters>
```

8.1.4.5.1 Data Component Union

The *DataComponentOrObservable* group provides three choices to serve as process inputs, outputs, and parameters. These include *swe:AbstractDataComponent*, *sml:ObservableProperty*, and *sml:DataInterface*. These will each be discussed further in the following sections.

The XML snippet defining the *DataComponentOrObservable* group is given below:

```
<group name="DataComponentOrObservable">
    <choice>
        <element ref="swe:AbstractDataComponent">
            <annotation>
                <documentation>
                    A single digital number (DN) or aggregate of DNs that represent the value of some property.
                    Single data components can be of type Quantity, Count, Category, Boolean, Text, or Time; these
                    can be aggregated in records, arrays, vector, and matrices.
                </documentation>
            </annotation>
        </element>
        <element ref="sml:ObservableProperty">
            <annotation>
                <documentation>
                    A physical property of the environment that can be observed by an appropriate detector (e.g.
                    temperature, pressure, etc.); Typically,an ObservableProperty serves as the input of a detector
                    and the output of an actuator.
                </documentation>
            </annotation>
        </element>
        <element ref="sml:DataInterface">
            <annotation>
                <documentation>
                    A data interface serves as an intermediary between the pure digital domain and the physical
                    domain where DN are encoded into a format and perhaps transmitted through physical
                    connections using some well-defined protocol. The DataInterface element allows one to define
                    the components, semantics, encoding, connections, and protocol at an input, output, or parameter
                    port.
                </documentation>
            </annotation>
        </element>
    </choice>
</group>

<complexType name="DataComponentOrObservablePropertyType">
    <sequence minOccurs="0">
        <group ref="sml:DataComponentOrObservable"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

8.1.4.5.2 ObservableProperty

The *ObservableProperty* element is an XML encoding of the model defined in Clause 7.2.1 and discussed more in Clause 7.2.3.1. The *ObservableProperty* is used to represent an observable property or state in the environment. It is often used as an input field (e.g. stimulus) for a detector or the output field (e.g. action) of an actuator.
The XML snippet defining *ObservableProperty* and its associated complex types is shown below:

```
<element name="ObservableProperty" type="sml:ObservablePropertyType"
    substitutionGroup="swe:AbstractSWEIdentifiable">
    <annotation>
        <documentation>
            A physical property that can be observed and possibly measured (e.g. temperature, color, position). An
            ObservableProperty has unambiguous definition, but does not have units of measure.
        </documentation>
    </annotation>
</element>

<complexType name="ObservablePropertyType">
    <complexContent>
        <extension base="swe:AbstractSWEIdentifiableType">
            <attribute name="definition" type="anyURI" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="ObservablePropertyPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ObservableProperty"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

The *ObservableProperty* element requires a URL as the value of its definition attribute, which should resolve to a definition for the observable within an online dictionary or ontology.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/observable-definition
Req 58. The *ObservableProperty* element shall include the *definition* attribute that shall take as its value, a resolvable URL referencing the definition of the observable within an online dictionary or ontology.

An minimal example of the use of *ObservableProperty* as a value for an input property is given below:

```
<sml:input name="temperature">
    <sml:ObservableProperty
        definition="http://sensorml.com/ont/swe/property/AtmosphericTemperature"/>
</sml:input>
```

8.1.4.5.3 SWE Common Data

SWE Common data types support the description and encoding of determined values, those measured or calculated by a physical or computational process. In addition to the those properties described in ObservableProperty, SWE Common data types can also support other properties such as units of measure, constraints, quality indicators, and values. It is highly recommended that one become familiar with the SWE Common Data specification in OGC 08-094.

In addition to simple data types such as *Quantity*, *Count*, *Boolean*, *Category*, *Time*, and *Text*, SWE Common Data also supports several aggregate data types, such as *DataRecord*, *DataArray*, *Vector*, and *Matrix*. These allow appropriate grouping of data components, such as a *DataRecord* including an instantaneous snapshot of the atmosphere (e.g. temperature, pressure, wind direction, and wind speed), a *DataArray* providing the axes definitions and values for a calibration curve, a *Vector* providing the GPS location (e.g. latitude, longitude, and altitude) of a dynamic platform, or a *Matrix* providing the covariance of a set of measurements.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/data-record
Req 59. Input, output, and parameter data components that are dependent on one another or represent the state at a given time, shall be surrounded by an appropriate aggregate data component

The following example for a wind chill calculation process uses only SWE Common *Quantity* elements for inputs and outputs. Note that the input fields are surrounded by a *swe:DataRecord* element since the temperature and wind speed fields express the condition of the environment at a given time instance.

```
<!--================ -->
<!--          Inputs          -->
<!-- ================ -->
<sml:inputs>
   <sml:InputList>
      <sml:input name="process_inputs">
         <swe:DataRecord>
            <swe:field name="temperature">
               <swe:Quantity definition="http://sweet.jpl.nasa.gov/2.2/quanTemperature.owl#Temperature">
                  <swe:uom code="cel"/>
               </swe:Quantity>
            </swe:field>
            <swe:field name="windSpeed">
               <swe:Quantity definition="http://sweet.jpl.nasa.gov/2.2/quanSpeed.owl#WindSpeed">
                  <swe:uom code="m/s"/>
               </swe:Quantity>
            </swe:field>
         </swe:DataRecord>
      </sml:input>
   </sml:InputList>
</sml:inputs>
```

```
<!-- ================ -->
<!--          Outputs          -->
<!-- ================ -->
<sml:outputs>
   <sml:OutputList>
      <sml:output name="windChill">
         <swe:Quantity definition="http://sweet.jpl.nasa.gov/2.2/quanTemperature.owl#WindChill">
            <swe:uom code="cel"/>
         </swe:Quantity>
      </sml:output>
   </sml:OutputList>
</sml:outputs>
```

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/vector-use
Req 60. A *swe:Vector* shall be used for any input, output, or parameter value that requires position to be specified relative to a specific axis of a reference frame.

An example of position output provided as output of a system is given below. This example uses both Vector and DataRecord elements to group the fields appropriately.

```
<sml:output>
    <swe:DataRecord>
      <swe:field name="location">
        <swe:Vector
             definition="http://sensorml.com/ont/swe/property/SensorLocation"
             referenceFrame="http://www.opengis.net/def/crs/EPSG/6.7/4979"
             localFrame="#SENSOR_FRAME">
          <swe:coordinate name="Lat">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Latitude" axisID="Lat">
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="Lon">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Longitude" axisID="Long">
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="Alt">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Altitude" axisID="Alt">
              <swe:uom code="m"/>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:field>
      <swe:field name="orientation">
        <swe:Vector
             definition="http://sensorml.com/def/property/SensorOrientation"
             referenceFrame="http://www.opengis.net/def/crs/NED"
             localFrame="#SENSOR_FRAME">
          <swe:coordinate name="TrueHeading">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/TrueHeading" axisID="Z">
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="Pitch">
```

```
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Pitch" axisID="Y">
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:field>
    </swe:DataRecord>
  </sml:output>
```

8.1.4.5.4 Data Interface

The *DataInterface* element is an XML encoding of the model defined in Clause 7.10. *DataInterface* describes the data components as described in the previous clause, but also provides a description of the data encoding of a datastream flowing to or from a process interface as well as a description of interface characteristics itself (e.g. the communication protocols and perhaps physical nature as outlined in the OSI interface stack).

The XML snippet for the *DataInterface* element and its associated complex types is shown below:

```
<element name="DataInterface" type="sml:DataInterfaceType" substitutionGroup="swe:AbstractSWEIdentifiable">
    <annotation>
        <documentation>
            The DataInterface description provides information sufficient for "plug-and-play" access to and parsing of
            the data stream or file at the particular IO port.
        </documentation>
    </annotation>
</element>

<complexType name="DataInterfaceType">
    <complexContent>
        <extension base="swe:AbstractSWEIdentifiableType">
            <sequence>
                <element name="data" type="swe:DataStreamPropertyType">
                    <annotation>
                        <documentation>
                            The definition of the digital data components and encoding accessed through the data
                            interface.
                        </documentation>
                    </annotation>
                </element>
                <element name="interfaceParameters" type="swe:DataRecordPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            A set of property values that define the type and configuration of a data interface (e.g. the
                            port settings of an RS232 interface).
                        </documentation>
                    </annotation>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="DataInterfacePropertyType">
    <sequence minOccurs="0">
        <element ref="sml:DataInterface"/>
    </sequence>
```

```
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
  </complexType>
```

The *DataInterface* essentially consists of a required *data* property, which takes a SWE Common *DataStream* element as its value, and an optional *interfaceParameters* property that defines the interface as a set of fields within a SWE Common *DataRecord*. This specification does not define the specific fields required to define the interface, but it is expected that one or more profiles will be defined for common interface definitions.
In the following example, the *values* property reference a RESTful resource (e.g. a daily record of weather) available on the web through html protocol. This example is for a network of weather sensors; thus the network stream includes station ID and location as part of the data components. In this case, a description of the InterfaceParameters is neither required nor useful.

```
<sml:outputs>
  <sml:OutputList>
    <sml:output name="sensorNetworkStream">
      <sml:DataInterface>
        <sml:data>
          <swe:DataStream>
              <swe:description>
                  This stream is for a sensor network where all output are homogeneous;
                  The station ID and location is provided with each reading;
               </swe:description>
            <swe:elementType name="weather_data">
              <swe:DataRecord>
                <swe:label>Atmospheric Conditions</swe:label>
                <swe:field name="time">
                  <swe:Time
                    definition="http://sensorml.com/ont/swe/property/SamplingTime">
                    <swe:uom
                      xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
                  </swe:Time>
                </swe:field>
                <swe:field name="sensorID">
                  <swe:Text
                    definition="http://sensorml.com/ont/swe/property/SensorID"/>
                </swe:field>

                <swe:field name="location">
                  <swe:Vector
                    definition="http://sensorml.com/ont/swe/property/PlatformLocation"
                    referenceFrame="http://www.opengis.net/def/crs/EPSG/0/4979">
                    <swe:coordinate name="lat">
                      <swe:Quantity
                        definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Latitude"
                        axisID="Lat">
                        <swe:uom code="deg"/>
                      </swe:Quantity>
                    </swe:coordinate>
                    <swe:coordinate name="lon">
                      <swe:Quantity
                        definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Longitude"
                        axisID="Long">
                        <swe:uom code="deg"/>
                      </swe:Quantity>
                    </swe:coordinate>
                    <swe:coordinate name="alt">
                      <swe:Quantity
```

```
                    definition="http://sweet.jpl.nasa.gov/2.0/spaceExtent.owl#Altitude"
                    axisID="h">
                    <swe:uom code="m"/>
                  </swe:Quantity>
                </swe:coordinate>
              </swe:Vector>
            </swe:field>

            <swe:field name="temp">
              <swe:Quantity
                definition="http://mmisw.org/ont/cf/parameter/air_temperature">
                <swe:uom code="Cel"/>
              </swe:Quantity>
            </swe:field>
            <swe:field name="pressure">
              <swe:Quantity
                definition="http://mmisw.org/ont/cf/parameter/barometric_pressure">
                <swe:uom code="kPa"/>
              </swe:Quantity>
            </swe:field>
            <swe:field name="wind_speed">
              <swe:Quantity
                definition="http://mmisw.org/ont/cf/parameter/wind_speed">
                <swe:uom code="km/h"/>
              </swe:Quantity>
            </swe:field>
            <swe:field name="wind_dir">
              <swe:Quantity
                definition="http://mmisw.org/ont/cf/parameter/wind_to_direction">
                <swe:uom code="deg"/>
              </swe:Quantity>
            </swe:field>
          </swe:DataRecord>
        </swe:elementType>
        <swe:encoding>
          <swe:TextEncoding tokenSeparator="," blockSeparator=""/>
        </swe:encoding>

        <swe:values xlink:href="rtp://mySensors.com:4356/76455"/>

      </swe:DataStream>
    </sml:data>
  </sml:DataInterface>
</sml:output>
```

The returned resource "weatherNetwork201.txt" might return the most recent reading from the network and look like:

```
2009-05-23T19:36:15Z,urn:myNet:stations:76455,35.4,135.6,5.0,25.4,100.3,7.31,270.8
2009-05-23T19:36:15Z,urn:myNet:stations:55577,34.1,138.9,4.1,25.5,100.5,7.54,271.4
2009-05-23T19:38:15Z,urn:myNet:stations:85643,43.9,141.0,3.8,25.7,100.1,7.44,260.2
2009-05-23T19:39:15Z,urn:myNet:stations:22298,46.7,140.0,1.2,26.5,100.6,7.30,271.9
2009-05-23T19:40:15Z,urn:myNet:stations:92675,43.1,131.0,6.7,25.5,100.2,7.54,271.0
```

The following example also references an external source for the values of the stream, but instead of a static resource, it references a Real-Time-Protocol (RTP) server that will continue to send real-time measurements until the client disconnects.

```
<sml:output name="gammaRadiation">
  <sml:DataInterface>

    <!-- data description -->
    <sml:data>
```

```
        <swe:DataStream>
          <swe:elementType name="gamaRadiation">
            <swe:DataRecord>
              <swe:field name="averageDose">
                <swe:Quantity definition="http://sensorml.com/ont/swe/property/DoseRate.html">
                  <swe:label>Average Dose of Gamma Radiation</swe:label>
                  <swe:uom code="uR/min"/>
                </swe:Quantity>
              </swe:field>
              <swe:field name="InstantaneousDose">
                <swe:Quantity definition="http://sensorml.com/ont/swe/property/RadiationDose">
                  <swe:label>Instantaneous Dose of Gamma Radiation</swe:label>
                  <swe:uom code="uR"/>
                </swe:Quantity>
              </swe:field>
            </swe:DataRecord>
          </swe:elementType>

          <!-- encoding description -->
          <swe:encoding>
            <swe:TextEncoding tokenSeparator="," blockSeparator=" "/>
          </swe:encoding>

          <!-- a Real-Time-Protocol (RTP) server that continues to stream real time measurements -->
          <swe:values xlink:href="rtp://myServer.com:4563/sensor/02080"/>

        </swe:DataStream>
      </sml:data>
    </sml:DataInterface>
  </sml:output>
```

Once one establishes connection to such a server, the server would then begin streaming over real-time values for the data as they become available. Such a stream might look like:

```
7248,26.3 7248,26.4 7250,26.6 7251,28.3 ...   [continues until disconnected]
```

Since SWE Common also supports binary encodings, the same sensor system could describe binary data, as in the current example:

```
  <sml:output name="gammaRadiation">
    <sml:DataInterface>

      <!-- data description -->
      <sml:data>
        <swe:DataStream>
          <swe:elementType name="gamaRadiation">
            <swe:DataRecord>
              <swe:field name="averageDose">
                <swe:Quantity definition="http://sensorml.com/ont/swe/property/DoseRate.html">
                  <swe:label>Average Dose of Gamma Radiation</swe:label>
                  <swe:uom code="uR/min"/>
                </swe:Quantity>
              </swe:field>
              <swe:field name="InstantaneousDose">
                <swe:Quantity definition="http://sensorml.com/ont/swe/property/RadiationDose">
                  <swe:label>Instantaneous Dose of Gamma Radiation</swe:label>
                  <swe:uom code="uR"/>
                </swe:Quantity>
              </swe:field>
            </swe:DataRecord>
          </swe:elementType>
```

```
            <swe:encoding>
                <swe:BinaryEncoding byteEncoding="raw" byteOrder="bigEndian">
                    <swe:member>
                        <swe:Component ref="averageDose"
                            dataType="http://www.opengis.net/def/dataType/OGC/0/float32"/>
                    </swe:member>
                    <swe:member>
                        <swe:Component ref="instantaneousDose"
                            dataType="http://www.opengis.net/def/dataType/OGC/0/float32"/>
                    </swe:member>
                </swe:BinaryEncoding>
            </swe:encoding>

            <swe:values xlink:href="rtp://myServer.com:4563/sensor/02080"/>

        </swe:DataStream>
    </sml:data>
  </sml:DataInterface>
</sml:output>
```

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/data-stream-url
Req 61. When the values of a *DataStream* can be referenced by a resolvable URL, that URL shall be the value of of the *xlink:href* attribute in the *value* property of the *DataStream*.

Using the SWE Common Data element, *DataChoice*, it is also possible to define and stream a collection of disparate packets within the same *DataStream*. According to the SWE Common Data specification, the structure of each packet should be defined as a separate *item* within the *DataChoice* object. The subsequent values of each packet should be preceded by the item name, as in the example XML and data stream below:

```
<sml:output name="tempAndWind">
  <sml:DataInterface>
    <sml:data>
      <swe:DataStream>
        <swe:elementType name="message">
          <swe:DataChoice>

            <!-- packet definition 1 - temperature -->
            <swe:item name="TEMP">
              <swe:DataRecord>
                <swe:label>Temperature Measurement</swe:label>
                <swe:field name="time">
                  <swe:Time definition="http://sensorml.com/ont/swe/property/SamplingTime">
                    <swe:uom  xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
                  </swe:Time>
                </swe:field>
                <swe:field name="temp">
                  <swe:Quantity definition="http://mmisw.org/ont/cf/parameter/air_temperature">
                    <swe:uom code="Cel"/>
                  </swe:Quantity>
                </swe:field>
              </swe:DataRecord>
```

```
            </swe:item>

            <!-- packet definition 2 - wind -->
            <swe:item name="WIND">
              <swe:DataRecord>
                <swe:label>Wind Measurement</swe:label>
                <swe:field name="time">
                  <swe:Time  definition="http://sensorml.com/ont/swe/property/SamplingTime">
                    <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
                  </swe:Time>
                </swe:field>
                <swe:field name="wind_speed">
                  <swe:Quantity definition="http://mmisw.org/ont/cf/parameter/wind_speed">
                    <swe:uom code="km/h"/>
                  </swe:Quantity>
                </swe:field>
                <swe:field name="wind_dir">
                  <swe:Quantity definition="http://mmisw.org/ont/cf/parameter/wind_to_direction">
                    <swe:uom code="deg"/>
                  </swe:Quantity>
                </swe:field>
              </swe:DataRecord>
            </swe:item>
          </swe:DataChoice>
        </swe:elementType>

        <swe:encoding>
          <swe:TextEncoding tokenSeparator="," blockSeparator=""/>
        </swe:encoding>

        <swe:values xlink:href="rtp://mySensors.com:4356/76455"/>

      </swe:DataStream>
    </sml:data>
  </sml:DataInterface>
</sml:output>
```

With an example streaming values looking like:

```
TEMP,2009-05-23T19:36:15Z,25.5
TEMP,2009-05-23T19:37:15Z,25.6
WIND,2009-05-23T19:37:17Z,56.3,226.3
TEMP,2009-05-23T19:38:15Z,25.5
WIND,2009-05-23T19:38:16Z,58.4,225.1
```

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/multiplexed-data-stream
Req 62. When the values of a *DataStream* consist of a series of multiplexed packets, the value of the *elementType* property shall be specified using the SWE Common Data *DataChoice* element.

In addition to online interfaces to data, *DataInterface* can also support physical interfaces such as RS232 or USB. In such cases, the *value* property is ignored, and the *interfaceParameters* property of the *DataInterface* element should be used to specify the interface characteristics.

An example of an interface description is given below. Since interface description profiles will be created under separate documents, the following example is informative, not normative.

```
<sml:interfaceParameters>
    <swe:DataRecord definition="http://sensorml.com/ont/swe/property/SerialPortSettings">
        <swe:field name="portType">
            <swe:Category definition="http://sensorml.com/ont/swe/property/SerialPortType">
                <swe:label>Port Type</swe:label>
                <swe:value>RS232</swe:value>
            </swe:Category>
        </swe:field>
        <swe:field name="portNumber">
            <swe:Count definition="http://sensorml.com/ont/swe/property/PortNumber">
                <swe:label>Port Number</swe:label>
                <swe:value>0</swe:value>
            </swe:Count>
        </swe:field>
        <swe:field name="baudRate">
            <swe:Count definition="http://sensorml.com/ont/swe/property/BaudRate">
                <swe:label>Baud Rate</swe:label>
                <swe:value>9600</swe:value>
            </swe:Count>
        </swe:field>
        <swe:field name="bits">
            <swe:Count definition="http://sensorml.com/ont/swe/property/DataBitSize">
                <swe:label>Data Bits</swe:label>
                <swe:value>8</swe:value>
            </swe:Count>
        </swe:field>
        <swe:field name="parity">
            <swe:Category definition="http://sensorml.com/ont/swe/property/DataParity">
                <swe:label>Parity</swe:label>
                <swe:value>N</swe:value>
            </swe:Category>
        </swe:field>
        <swe:field name="stopBits">
            <swe:Count definition="http://sensorml.com/ont/swe/property/StopBits">
                <swe:label>Stop Bits</swe:label>
                <swe:value>1</swe:value>
            </swe:Count>
        </swe:field>
    </swe:DataRecord>
</sml:interfaceParameters>
```

8.1.4.6 Modes

The *AbstractModes* element provides a base class from which will be derived a concrete *Modes* class in the higher-level conformance class specified by the Requirements Class: Configurable Process Schema in Clause 8.6.

The XML Schema implementation of AbstractModes and its related complex type is given below:

```
<element name="AbstractModes" type="sml:AbstractModesType" substitutionGroup="swe:AbstractSWE"/>

<complexType name="AbstractModesType">
    <complexContent>
        <extension base="swe:AbstractSWEType"/>
```

```
    </complexContent>
</complexType>

<complexType name="AbstractModesPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:AbstractModes"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

8.2 Requirements Class: Simple Process Schema

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/xml/simple-process	
Target Type	XML Instance
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process

XML Schema elements and types defined in the "*simple_process.xsd*" schema file implement all classes defined respectively in the "simple-process" UML package defined in Clause 7.3.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/simple-process/schema-valid
Req 63. The XML instance shall be valid with respect to the XML grammar defined in the "simple_process.xsd", as well as satisfy all Schematron patterns defined in "simple_process.sch".

The following XML snippet provides the appropriate header and import statements for the simple_process.xsd:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:sml="http://www.opengis.net/sensorML/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:swe="http://www.opengis.net/swe/2.0"
    targetNamespace="http://www.opengis.net/sensorML/2.0" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <annotation>
        <documentation>
            The elements of a concrete simple non-physical process derived from the core process
            model.
        </documentation>
    </annotation>
    <import namespace="http://www.opengis.net/gml/3.2"
        schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
    <import namespace="http://www.opengis.net/swe/2.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/2.0/swe.xsd"/>
    <include schemaLocation="core.xsd"/>
```

8.2.1 Simple Process

The "*SimpleProcesss*" element is the XML schema implementation of the "*SimpleProcess*" UML class defined in clause 7.3.1. The schema snippet for this element and its corresponding complex type is shown below:

```
    <element name="SimpleProcess" type="sml:SimpleProcessType" substitutionGroup="sml:AbstractProcess"/>

    <complexType name="SimpleProcessType">
        <complexContent>
```

```
        <extension base="sml:AbstractProcessType">
            <sequence>
                <element name="method" type="sml:ProcessMethodPropertyType" minOccurs="0" maxOccurs="1"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="SimpleProcessPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:SimpleProcess"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

The *SimpleProcess* element and complex type is derived from *AbstractProcess* and adds a *method* property to the properties inherited from *AbstractProcess*. The *method* property takes a *ProcessMethod* element as its value.

8.2.2 Process Method

The "*ProcessMethod*" element is the XML schema implementation of the "*ProcessMethod*" UML class defined in clause 7.3.2. The schema snippet for this element and its corresponding complex type is shown below:

```
<element name="ProcessMethod" type="sml:ProcessMethodType"
    substitutionGroup="swe:AbstractSWEIdentifiable">
    <annotation>
        <documentation>
            The method describes (as an algorithm or text) how the process takes the input and parameter values
            and generates output values.
        </documentation>
    </annotation>
</element>

<complexType name="ProcessMethodType">
    <complexContent>
        <extension base="swe:AbstractSWEIdentifiableType">
            <sequence>
                <element name="algorithm" minOccurs="0" maxOccurs="unbounded">
                    <complexType>
                        <sequence>
                            <element ref="sml:AbstractAlgorithm"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ProcessMethodPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ProcessMethod"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>

<element name="AbstractAlgorithm" type="sml:AbstractAlgorithmType" abstract="true"
    substitutionGroup="gml:AbstractObject"/>
```

```
<complexType name="AbstractAlgorithmType" abstract="true">
    <sequence/>
    <attribute ref="gml:id" use="optional"/>
</complexType>

<complexType name="AbstractAlgorithmPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:AbstractAlgorithm"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

The *ProcessMethod* element is derived from *SWEIdentifiableType* and thus supports the *swe:identifier*, *swe:label*, and *swe:description* properties, as well as the *swe:extension* property, which supports the ability to specify community-specific XML elements for describing the method.
Furthermore, *ProcessMethod* has an *algorithm* property that takes an *AbstractAlgorithm* element as its value. It is anticipated that profiles will be specified in separate specifications that will support algorithms descriptions through schema such as MathML.

8.3 Requirements Class: Aggregate Process Schema

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process	
Target Type	XML Instance
Dependency	**http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process**
Dependency	**http://www.opengis.net/spec/sensorml/2.0/req/xml/simple-process**

XML Schema elements and types defined in the "*aggregate_process.xsd*" schema file implement all classes defined respectively in the "aggregate-process" UML package defined in Clause 7.4.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/schema-valid
Req 64. The XML instance shall be valid with respect to the XML grammar defined in the "aggregate_process.xsd", as well as satisfy all Schematron patterns defined in "aggregate_process.sch".

The following XML snippet provides the appropriate header and import statements for the simple_process.xsd:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:sml="http://www.opengis.net/sensorML/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:swe="http://www.opengis.net/swe/2.0"
    targetNamespace="http://www.opengis.net/sensorML/2.0" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <annotation>
        <documentation>
            The elements of a concrete aggregate non-physical process derived from the core process
            model.
        </documentation>
    </annotation>
    <import namespace="http://www.opengis.net/gml/3.2"
        schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
    <import namespace="http://www.opengis.net/swe/2.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/2.0/swe.xsd"/>
    <include schemaLocation="simple_process.xsd"/>
```

8.3.1 Aggregate Process

The "*AggregateProcess*" element is the XML schema implementation of the "*AggregateProcess*" UML class defined in clause 7.4.1. The schema snippet for this element and its corresponding complex type is shown below:

```
<element name="AggregateProcess" type="sml:AggregateProcessType" substitutionGroup="sml:AbstractProcess">
    <annotation>
        <documentation>
            A process that consist of a collection of linked component processes resulting in a specified
            output.
        </documentation>
    </annotation>
</element>

<complexType name="AggregateProcessType">
    <complexContent>
        <extension base="sml:AbstractProcessType">
            <sequence>
                <element name="components" type="sml:ComponentListPropertyType"
                    minOccurs="0" maxOccurs="1"/>
                <element name="connections" type="sml:ConnectionListPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            The explicit definition of data links between outputs, inputs, and parameters of the
                            components within an aggregate process.
                        </documentation>
                    </annotation>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="AggregateProcessPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:AggregateProcess"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

The *AggregateProcess* element and complex type is derived from *AbstractProcess* and adds *components* and *connections* properties to the properties inherited from *AbstractProcess*. The *components* and *connections* properties take *ComponentList* and *ConnectionList* as their values, respectively.

The following two examples show the same aggregate process. A graphic of this "Slice and Clip" *AggregateProcess* is shown in Figure 8.1. In this diagram, inputs to the aggregate process and its components are shown on the left of the object, while outputs are on the right and parameters are on top. In this example, the parameter values are set and constant, as shown by the values assigned them. However, parameters can be variable and supplied by connections as can inputs.

This diagram and the following SensorML instances illustrate that an AggregateProcess can have multiple components and connections that will be discussed in more detail in the following Sections 8.3.2 and 8.3.3. Notice that inputs, outputs, and parameters can be defined for the aggregate process as well as for the individual components.

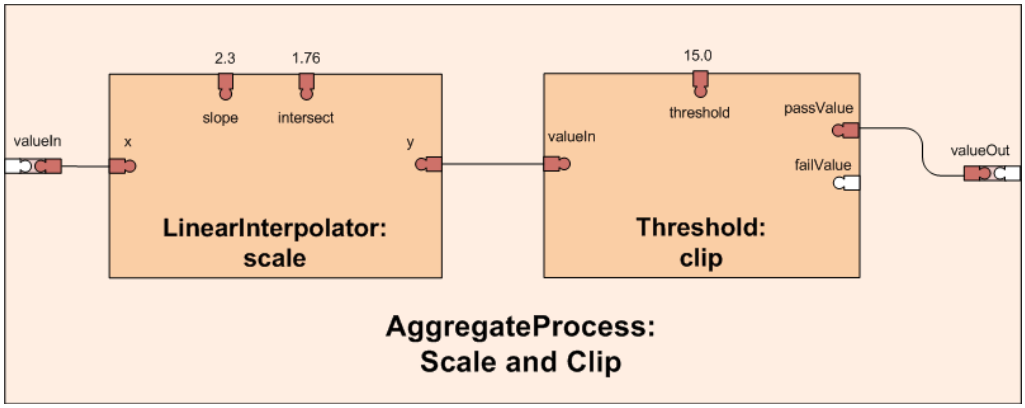


Figure 8.1. A diagram showing the components and connections of the "Scale and Clip" AggregateProcess.

The first example provides a complete definition of the component processes and the setting of parameter values inline.

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:AggregateProcess gml:id="scaleAndClip01"
   xmlns:sml="http://www.opengis.net/sensorML/2.0"
   xmlns:swe="http://www.opengis.net/swe/2.0"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xsi:schemaLocation="http://www.opengis.net/sensorML/2.0 http://schemas.opengis.net/sensorml/2.0/SensorML.xsd "
   definition="http://sensors.ws/process/linearInterpolator">

   <!-- ============================== -->
   <!--       Descriptions             -->
   <!-- ============================== -->
   <gml:description>
         A simple aggregate process that scales according to linear equation y = 2.3x + 1.76 and then clips if below 15.0
         In this example all processes are defined inline with no configuration settings. Parameter values are set inline.
     </gml:description>
   <gml:identifier codeSpace="uid">urn:myCompany:swe:process:scaleAndClip01</gml:identifier>
   <gml:name>Scale and Clip Aggregate Process 01</gml:name>

   <!-- ============================== -->
   <!--          Aggregate Process Inputs          -->
   <!-- ============================== -->
   <sml:inputs>
      <sml:InputList>
         <sml:input name="valueIn">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/DN">
               <swe:uom code="any"/>
            </swe:Quantity>
         </sml:input>
      </sml:InputList>
   </sml:inputs>

   <!-- ============================== -->
   <!--          Aggregate Process Outputs          -->
   <!-- ============================== -->
   <sml:outputs>
      <sml:OutputList>
```

```
        <sml:output name="valueOut">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/DN">
            <swe:uom code="any"/>
          </swe:Quantity>
        </sml:output>
      </sml:OutputList>
    </sml:outputs>

    <!-- ========================================= -->
    <!--        Aggregate process components declared          -->
    <!-- ========================================= -->
    <sml:components>
        <sml:ComponentList >

            <!-- Component 1 - Linear Interpolator -->
            <sml:component name="scale">
                <sml:SimpleProcess gml:id="linearInterpolator01"
                    definition="http://sensors.ws/process/linearInterpolator">
                    <!-- ============================== -->
                    <!--      Linear Interpolator Descriptions          -->
                    <!-- ============================== -->
                    <gml:description>A linear interpolator based on equation y = mx + b </gml:description>
                    <gml:identifier codeSpace="uid">urn:myCompany:process:8755d73ab</gml:identifier>
                    <gml:name>Linear Equation 01</gml:name>
                    <!-- ============================== -->
                    <!--          Linear Interpolator Inputs          -->
                    <!-- ============================== -->
                    <sml:inputs>
                        <sml:InputList>
                            <sml:input name="x">
                                <swe:Quantity definition="http://sensorml.com/ont/swe/property/XCoordinate">
                                    <swe:uom code="any"/>
                                </swe:Quantity>
                            </sml:input>
                        </sml:InputList>
                    </sml:inputs>
                    <!-- ============================== -->
                    <!--          Linear Interpolator Outputs          -->
                    <!-- ============================== -->
                    <sml:outputs>
                        <sml:OutputList>
                        <!-- scaled output value -->
                            <sml:output name="y">
                                <swe:Quantity definition="http://sensorml.com/ont/swe/property/DN">
                                    <swe:uom code="any"/>
                                </swe:Quantity>
                            </sml:output>
                        </sml:OutputList>
                     </sml:outputs>
                    <!-- ============================== -->
                    <!--       Linear Interpolator Parameters       -->
                    <!-- ============================== -->
                    <sml:parameters>
                        <sml:ParameterList>
                            <sml:parameter name="slope-intercept">
                                <swe:DataRecord>
                                    <swe:field name="slope">
                                        <swe:Quantity definition="http://sensorml.com/ont/swe/property/LinearSlope">
                                            <swe:uom code="any"/>
                                            <!-- slope value set inline -->
                                            <swe:value>2.3</swe:value>
                                        </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="intercept">
                                        <swe:Quantity
                                            definition="http://sensorml.com/ont/swe/property/LinearAxisIntercept">
```

```
                                    <!-- y-intercept value set inline -->
                                    <swe:uom code="any"/>
                                    <swe:value>1.76</swe:value>
                                </swe:Quantity>
                            </swe:field>
                        </swe:DataRecord>
                    </sml:parameter>
                </sml:ParameterList>
            </sml:parameters>
        </sml:SimpleProcess>
    </sml:component>


    <!-- Component 2 - Threshold clipper -->
    <sml:component name="clip">
        <sml:SimpleProcess gml:id="thresholdClipper"
            definition="http://sensors.ws/process/thresholdClipper">
            <!-- ============================== -->
            <!--     Threshold Clipper  Descriptions          -->
            <!-- ============================== -->
            <gml:description>
                A process that clips anything below threshold;
                values higher than threshold to passValue output;
                Values below threshold sent to failValue output</gml:description>
            <gml:identifier codeSpace="uid">urn:myCompany:swe:process:65d74a65c</gml:identifier>
            <gml:name>Threshold Clipper  01</gml:name>
            <!-- ============================== -->
            <!--           Threshold Clipper Inputs          -->
            <!-- ============================== -->
            <sml:inputs>
                <sml:InputList>
                    <sml:input name="valueIn">
                        <swe:Quantity definition="http://sensorml.com/ont/swe/property/DN">
                            <swe:uom code="any"/>
                        </swe:Quantity>
                    </sml:input>
                </sml:InputList>
            </sml:inputs>
            <!-- ============================== -->
            <!--          Threshold Clipper Outputs          -->
            <!-- ============================== -->
            <sml:outputs>
                <sml:OutputList>
                <!-- output for values that pass -->
                    <sml:output name="passValue">
                        <swe:Quantity definition="http://sensorml.com/ont/swe/property/PassValue">
                            <swe:uom code="any"/>
                        </swe:Quantity>
                    </sml:output>
                    <!-- output for values that fail -->
                    <sml:output name="failValue">
                        <swe:Quantity definition="http://sensorml.com/ont/swe/property/FailValue">
                            <swe:uom code="any"/>
                        </swe:Quantity>
                    </sml:output>
                </sml:OutputList>
             </sml:outputs>
            <!-- ============================== -->
            <!--      Threshold Clipper Parameters          -->
            <!-- ============================== -->
            <sml:parameters>
                <sml:ParameterList>
                    <sml:parameter name="threshold">
                        <swe:Quantity definition="http://sensorml.com/ont/swe/property/LowerThreshold">
                            <swe:uom code="any"/>
                            <!-- threshold value set inline -->
```

```
                                    <swe:value>15.0</swe:value>
                                </swe:Quantity>
                            </sml:parameter>
                        </sml:ParameterList>
                    </sml:parameters>
                </sml:SimpleProcess>
            </sml:component>

        </sml:ComponentList>
    </sml:components>

    <!-- ===================================== -->
    <!--        Aggregate process links declared          -->
    <!-- ===================================== -->
    <sml:connections>
        <sml:ConnectionList>

        <!-- Connect AggregateProcess input to LinearInterpolator (scale) input -->
            <sml:connection>
                <sml:Link>
                    <sml:source ref="inputs/valueIn"/>
                    <sml:destination ref="components/scale/inputs/x"/>
                </sml:Link>
            </sml:connection>

            <!-- Connect LinearInterpolator (scale) output to ThresholdClipper (clip) input -->
            <sml:connection>
                <sml:Link>
                    <sml:source ref="components/scale/outputs/y"/>
                    <sml:destination ref="components/clip/inputs/valueIn"/>
                </sml:Link>
            </sml:connection>

            <!-- Connect ThresholdClipper (clip) passValue output to AggregateProcess passValue output -->
            <sml:connection>
                <sml:Link>
                    <sml:source ref="components/clip/outputs/passValue"/>
                    <sml:destination ref="outputs/valueOut"/>
                </sml:Link>
            </sml:connection>

            <!-- Note: ThresholdClipper (clip) failValue output is ignored in this example -->

        </sml:ConnectionList>
    </sml:connections>
</sml:AggregateProcess>
```

Typically, however, aggregate processes will consist of well-defined, modular component processes that exist in a process repository and can be referenced and configured using the *typeOf* and *configuration* properties, respectively (discussed in subsequent Section 8.6). The second example utilizes components that are externally defined but are referenced and configured within the aggregate process description.

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:AggregateProcess gml:id="scaleAndClip06"
    xmlns:sml="http://www.opengis.net/sensorML/2.0"
    xmlns:swe="http://www.opengis.net/swe/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
xsi:schemaLocation="http://www.opengis.net/sensorML/2.0 http://schemas.opengis.net/sensorml/2.0/sensorML.xsd"
definition="http://sensors.ws/process/linearInterpolator">
<!-- ============================== -->
<!--       Descriptions             -->
<!-- ============================== -->
<gml:description>
  A simple aggregate process that scales according to linear equation y = 2.3x + 1.76 and then clips if below 15.0
  In this example all processes are defined externally and configured.
  </gml:description>
<gml:identifier codeSpace="uid">urn:myCompany:process:scaleAndClip06</gml:identifier>
<gml:name>Scale and Clip Aggregate Process 06</gml:name>
<!-- ============================== -->
<!--          Aggregate Process Inputs          -->
<!-- ============================== -->
<sml:inputs>
   <sml:InputList>
      <sml:input name="valueIn">
         <swe:Quantity definition="http://sensorml.com/ont/swe/property/DN">
            <swe:uom code="any"/>
         </swe:Quantity>
      </sml:input>
   </sml:InputList>
</sml:inputs>
<!-- ============================== -->
<!--          Aggregate Process Outputs          -->
<!-- ============================== -->
<sml:outputs>
   <sml:OutputList>
      <sml:output name="valueOut">
         <swe:Quantity definition="http://sensorml.com/ont/swe/property/DN">
            <swe:uom code="any"/>
         </swe:Quantity>
      </sml:output>
   </sml:OutputList>
 </sml:outputs>

 <!-- ======================================== -->
<!--        Aggregate process components declared           -->
<!-- ======================================== -->
<sml:components>
       <sml:ComponentList >

           <!-- Component 1 - Linear Interpolator -->
           <sml:component name="scale">
               <sml:SimpleProcess gml:id="linearInterpolatorConfigured"
                   definition="http://sensorml.com/ont/swe/process/LinearInterpolator">
                   <!-- ============================== -->
                   <!--       Linear Interpolator Descriptions          -->
                   <!-- ============================== -->
                   <gml:description>A linear interpolator with slope of 2.3 and intercept of 1.76</gml:description>
                   <gml:identifier codeSpace="uid">urn:myCompany:swe:process:09h57b21</gml:identifier>
                   <gml:name>Linear Equation 01 Configured</gml:name>
                    <sml:typeOf xlink:title="urn:net:swe:process:linearEquation01"
                       xlink:href="http://sensors.ws/processes/LinearInterpolator01"/>
                   <sml:configuration>
                       <sml:Settings>
                           <sml:setValue ref="parameters/slope-intercept/slope">2.3</sml:setValue>
                           <sml:setValue ref="parameters/slope-intercept/intercept">1.76</sml:setValue>
                       </sml:Settings>
                   </sml:configuration>
               </sml:SimpleProcess>
           </sml:component>


           <!-- Component 2 - Threshold clipper -->
           <sml:component name="clip">
```

```
                <sml:SimpleProcess gml:id="thresholdClipperConfigured"
                    definition="http://sensors.ws/process/thresholdClipper">
                    <!-- ============================== -->
                    <!--     Threshold Clipper  Descriptions          -->
                    <!-- ============================== -->
                    <gml:description>
                        A process that passes values of 15.0 and above to the passValue output;
                    </gml:description>
                    <gml:identifier codeSpace="uid">urn:myCompany:swe:process:0678b365a</gml:identifier>
                    <gml:name>Threshold Clipper  01 Configured</gml:name>
                    <sml:typeOf xlink:title="urn:net:swe:process:thresholdClip01"
                        xlink:href="http://sensors.ws/processes/ThresholdClipper01"/>
                    <sml:configuration>
                        <sml:Settings>
                            <sml:setValue ref="parameters/threshold">15.0</sml:setValue>
                        </sml:Settings>
                    </sml:configuration>
                </sml:SimpleProcess>
            </sml:component>

        </sml:ComponentList>
    </sml:components>

     <!-- ====================================== -->
    <!--       Aggregate process links declared                  -->
    <!-- ====================================== -->
    <sml:connections>
        <sml:ConnectionList>

        <!-- Connect AggregateProcess input to LinearInterpolator (scale) input -->
            <sml:connection>
                <sml:Link>
                    <sml:source ref="inputs/valueIn"/>
                    <sml:destination ref="components/scale/inputs/x"/>
                </sml:Link>
            </sml:connection>

            <!-- Connect LinearInterpolator (scale) output to ThresholdClipper (clip) input -->
            <sml:connection>
                <sml:Link>
                    <sml:source ref="components/scale/outputs/y"/>
                    <sml:destination ref="components/clip/inputs/valueIn"/>
                </sml:Link>
            </sml:connection>

            <!-- Connect ThresholdClipper (clip) passValue output to AggregateProcess passValue output -->
            <sml:connection>
                <sml:Link>
                    <sml:source ref="components/clip/outputs/passValue"/>
                    <sml:destination ref="outputs/valueOut"/>
                </sml:Link>
            </sml:connection>

            <!-- Note: ThresholdClipper (clip) failValue output is ignored in this example -->

        </sml:ConnectionList>
    </sml:connections>
</sml:AggregateProcess>
```

8.3.2 Components

The *components* property and the *ComponentList* elements are XML Schema implementations of the UML classes defined in clause 7.4. The *components* property

takes a *ComponentList* as its value. The *ComponentList component* property takes an *AbstractProcess* as its value. Thus any SensorML process can be a component of an aggregate process, whether that process is itself simple or aggregate, non-physical or physical.

The XML snippet for the *ComponentList* element and its corresponding complex types is shown below:

```
<element name="ComponentList" type="sml:ComponentListType" substitutionGroup="swe:AbstractSWE"/>

<complexType name="ComponentListType">
    <complexContent>
        <extension base="swe:AbstractSWEType">
            <sequence>
                <element name="component" minOccurs="1" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            A description of a component of the aggregate process. If by reference, the uniqueID of
                            the referenced process must be provided using the xlink:title attribute while the URL to
                            the process description must be provided by the xlink:href attribute.
                        </documentation>
                    </annotation>
                    <complexType>
                        <complexContent>
                            <extension base="sml:AbstractProcessPropertyType">
                                <attribute name="name" type="NCName" use="required"/>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ComponentListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ComponentList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

The component process can be described inline or be referenced through the xlink:href property. For a reference to an external process description, two pieces of information are required in order to reference a component process instance, the uniqueID of the process and a resolvable URL referencing the process description. These should be provided by the *xlink:title* and *xlink:href* attributes, respectively.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/component-reference
Req 65. When the value of the *component* property is "byReference", there shall be meaningful values for the *xlink:title* and *xlink:href* attributes. The uniqueID of the referenced process, given by its *gml:identifier* property, shall be the value of the *xlink:title* attribute, while a resolvable URL to the referenced process description shall be the value of the *xlink:href* attribute.

The examples in Section 8.3.1 show that the components of an aggregate process can be fully described inline, or be externally referenced processes that are configurable.

8.3.3 Connections

The *connections* property and the *ConnectionList* elements are XML Schema implementations of the UML classes defined in clause 7.4. The *connections* property takes a *ConnectionList* as its value. The *ConnectionList connection* property takes a *Link* as its value.

The XML snippet for the *ConnectionList* and *Link* elements and their corresponding complex types is shown below:

ConnectionList:

```
<element name="ConnectionList" type="sml:ConnectionListType" substitutionGroup="swe:AbstractSWE"/>

<complexType name="ConnectionListType">
    <complexContent>
        <extension base="swe:AbstractSWEType">
            <sequence>
                <element name="connection" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <sequence>
                            <element ref="sml:Link"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ConnectionListPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ConnectionList"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

Link:

```
<element name="Link" type="sml:LinkType" substitutionGroup="gml:AbstractObject"/>

<complexType name="LinkType">
    <sequence>
        <element name="source" type="sml:DataComponentRefPropertyType">
            <annotation>
                <documentation>The output from which the link originates.</documentation>
            </annotation>
        </element>
        <element name="destination" type="sml:DataComponentRefPropertyType">
            <annotation>
                <documentation>The input or parameter into which the data flows.</documentation>
            </annotation>
        </element>
    </sequence>
    <attribute ref="gml:id" use="optional"/>
</complexType>

<complexType name="LinkPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:Link"/>
```

```
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

From core.xsd:

```
<complexType name="DataComponentRefPropertyType">
    <attribute name="ref" type="sml:DataComponentPathPropertyType" use="required"/>
</complexType>

<simpleType name="DataComponentPathPropertyType">
    <restriction base="token">
        <pattern value="([a-zA-Z_][a-zA-Z0-9_\-\.]*)(/[a-zA-Z_][a-zA-Z0-9_\-\.]*)*"/>
    </restriction>
</simpleType>
```

8.3.3.1 Rules for Connections

The *Link* element may define a data connection between any properties of a process, but it typically connects inputs, outputs, and parameters. However, certain rules apply regarding sources for data (provided by the *source* property of the *Link* element) and destinations for data (provided by the *destination* property of the *Link* element).
Typical data flow is from an aggregate processes input to one or more component's input, from a component output to another component's input, or from a component's output to an output of the aggregate process.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/input-connection-restrictions
Req 66. The input of an enclosing aggregate process can connect to the input of one or more of its components; otherwise an input cannot connect to another input.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/output-connection-restrictions
Req 67. The output of a component can connect to an output of its enclosing aggregate process; otherwise an output cannot connect to another output.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/multiple-connections
Req 68. An output can connect to multiple destinations, but an input can only have one source connection.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/parameter-connection-restrictions
Req 69. A parameter can only be connected as a destination; a parameter cannot serve as a source.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/property-connection-restrictions
Req 70. Process properties other than inputs, outputs, or parameters can only serve as sources of data not destinations.

8.3.3.2 Rules for Path Designation

While the use of XPath was considered as a means of designating *Link* sources and destinations, this protocol was rejected on the grounds that its was too complicated for our needs and because it did not allow continues designation of paths for "byReference" objects. The designation of a path starts at the base element of the aggregate process and only uses property elements. Where the attribute name exists for an element, it is used; otherwise the element name is used.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/designating-link-paths
Req 71. The following rules shall be used for designating the source and destination paths within the Linked element. a) The path begins at the base of the current process b) The path includes only property elements (lowerCamelCase) and not value objects (UpperCamelCase) c) If the property has an attribute name and it has a value, then it shall be used; otherwise the unqualified name of the property element should be used d) The path can follow properties that are byReference

As an example, a snippet of the previous example is given below, highlighting the source path parts in yellow and the destination path parts in cyan. The subsequent example shows the appropriate source link path (in yellow) and the destination link path (in cyan).

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:AggregateProcess gml:id="scaleAndClip06"
```

```
… deleted for brevity sake …

<!-- ============================== -->
<!--          Aggregate Process Inputs          -->
<!-- ============================== -->
<sml:inputs>
   <sml:InputList>
      <sml:input name="valueIn">
         <swe:Quantity definition="http://sensorml.com/ont/swe/property/DN">
            <swe:uom code="any"/>
         </swe:Quantity>
      </sml:input>
   </sml:InputList>
</sml:inputs>

… deleted for brevity sake …


<sml:components>
      <sml:ComponentList >

         <!-- Component 1 - Linear Interpolator -->
         <sml:component name="scale">
            <sml:SimpleProcess gml:id="linearInterpolator01"
               definition="http://sensorml.com/ont/swe/process/LinearInterpolator">
               <!-- ============================== -->
               <!--      Linear Interpolator Descriptions          -->
               <!-- ============================== -->
               <gml:description>A linear interpolator based on equation y = mx + b </gml:description>
               <gml:identifier codeSpace="uid">urn:myCompany:process:8755d73ab</gml:identifier>
               <gml:name>Linear Equation 01</gml:name>
               <!-- ============================== -->
               <!--          Linear Interpolator Inputs          -->
               <!-- ============================== -->
               <sml:inputs>
                  <sml:InputList>
                     <sml:input name="x">
                        <swe:Quantity definition="http://sensorml.com/ont/swe/property/XCoordinate">
                           <swe:uom code="any"/>
                        </swe:Quantity>
                     </sml:input>
                  </sml:InputList>
               </sml:inputs>


… deleted for brevity sake …
```

```
<sml:connections>
      <sml:ConnectionList>

         <!-- Connect AggregateProcess input to LinearInterpolator (scale) input -->
         <sml:connection>
            <sml:Link>
               <sml:source ref="inputs/valueIn"/>
               <sml:destination ref="components/scale/inputs/x"/>
            </sml:Link>
         </sml:connection>

… deleted for brevity sake …
```

8.4 Requirements Class: Physical Component Schema

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component	
Target Type	XML Instance
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/xml/simple-process

XML Schema elements and types defined in the "*physical_component.xsd*" schema file implement all classes defined respectively in the "physical-component" UML package defined in Clause 7.6.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/schema-valid
Req 72. The XML instance shall be valid with respect to the XML grammar defined in the "physical_component.xsd", as well as satisfy all Schematron patterns defined in "physical_component.sch".

The following XML snippet provides the appropriate header and import statements for the physical_component.xsd:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:sml="http://www.opengis.net/sensorML/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:swe="http://www.opengis.net/swe/2.0"
    targetNamespace="http://www.opengis.net/sensorML/2.0" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <annotation>
        <documentation>
            The elements of a concrete simple physical process derived from the core process model.
        </documentation>
    </annotation>
    <import namespace="http://www.opengis.net/swe/2.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/2.0/swe.xsd"/>
    <import namespace="http://www.opengis.net/gml/3.2"
        schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
    <include schemaLocation="simple_process.xsd"/>
```

8.4.1 Abstract Physical Process

AbstractPhysicalProcess is derived from *AbstractProcess* and serves as the base class for all physical processes modelled and encoded in this specification..

The "*AbstractPhysicalProcess*" element is the XML schema implementation of the "*AbstractPhysicalProcess*" UML class defined in clause 7.6. The schema snippet for this element and its corresponding complex type is shown below.

```
<element name="AbstractPhysicalProcess" type="sml:AbstractPhysicalProcessType" abstract="true"
    substitutionGroup="sml:AbstractProcess">
    <annotation>
        <documentation>
            A physical process where the spatial and temporal state of the process is relevant.
        </documentation>
    </annotation>
</element>
<complexType name="AbstractPhysicalProcessType" abstract="true">
    <complexContent>
        <extension base="sml:AbstractProcessType">
            <sequence>
                <element name="attachedTo" type="gml:ReferenceType" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <appinfo>
                            <gml:targetElement>sml:AbstractPhysicalProcess</gml:targetElement>
                        </appinfo>
                        <documentation>
                            References the physical component or system (e.g. platform) to which to which this
                            component or system is attached.
                        </documentation>
                    </annotation>
                </element>
                <element name="localReferenceFrame" minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            A spatial reference frame of the physical component itself; this reference frame is
                            absolute and defines the relationship of the reference frame to the physical body of the
                            component; position of the component relates this reference frame to some external
                            reference frame. Note that units are specified in the position so they are not specified as
                            part of the SpatialFrame.
                        </documentation>
                    </annotation>
                    <complexType>
                        <sequence>
                            <element ref="sml:SpatialFrame"/>
                        </sequence>
                    </complexType>
                </element>
                <element name="localTimeFrame" minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            Supports local time reference frames such as "time past mission start". Note that units are
                            handled in timePosition so they are not specified in the TemporalFrame.
                        </documentation>
                    </an notation>
                    <complexType>
                        <sequence>
                            <element ref="sml:TemporalFrame"/>
                        </sequence>
                    </complexType>
                </element>
                <element name="position" type="sml:PositionUnionPropertyType"
                    minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            Provides positional information relating the component's spatial reference frame to an
                            external spatial reference frame. Positional information can be given by location, by full
                            body state, by a time-tagged trajectory, or by a measuring or computational process.
                        </documentation>
```

```
                    </annotation>
                </element>
                <element name="timePosition" type="swe:TimePropertyType"
                    minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            Provides Time positions typically reference a local time frame to an external time frame.
                            For example, a timer-start-time might be given relative to an "absolute" GPS time.
                        </documentation>
                    </annotation>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="AbstractPhysicalProcessPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:AbstractPhysicalProcess"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

AbstractPhysicalProcess element adds the following properties to the *AbstractProcess*: *attachedTo*, *localReferenceFrame*, *temporalReferenceFrame*, and *position*. These properties of *AbstractPhysicalProcess* will be discussed in more detail in the following subsections.

8.4.1.1 AttachedTo

The *attachedTo* property provides a reference to either a *PhysicalComponent* or *PhysicalSystem*. It provides a means for an attached component to reference back to the component or system to which it is attached. This is particularly useful when the movement of the parent affects the movement of the component.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/attached-to-target
Req 73. The *attachedTo* property shall provide a reference to an object of type *PhysicalComponent* or *PhysicalSystem*.

The same rules should apply as those for component references stated earlier.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/attached-to-reference
Req 74. The *attachedTo* property shall require meaningful values for the *xlink:title* and *xlink:href* attributes. The uniqueID of the referenced process, given by its *gml:identifier* property, shall be the value of the *xlink:title* attribute, while a resolvable URL to the reference process description shall be the value of the

***xlink:href* attribute.**

8.4.1.2 Local Reference Frame

The *localReferenceFrame* property takes a *SpatialFrame* as its value. The "*SpatialFrame*" element is an XML schema implementation of the "*SpatialFrame*" UML class defined in clause 7.6. The schema snippet for this element and its corresponding complex type is shown below.

```
<element name="SpatialFrame" type="sml:SpatialFrameType" substitutionGroup="swe:AbstractSWEIdentifiable">
    <annotation>
        <documentation>
            A general spatial Cartesian Reference Frame where the axes and origin will be defined textually relative
            to a physical component.
        </documentation>
    </annotation>
</element>

<complexType name="SpatialFrameType">
    <complexContent>
        <extension base="swe:AbstractSWEIdentifiableType">
            <sequence>
                <element name="origin" type="string">
                    <annotation>
                        <documentation>
                            A textual description of the origin of the reference frame relative to the physical device
                            (e.g. "the origin is at the point of attachment of the sensor to the platform").
                        </documentation>
                    </annotation>
                </element>
                <element name="axis" minOccurs="1" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            Axis with name attribute and a textual description of the relationship of the axis to the
                            physical device; the order of the axes listed determines their relationship according to the
                            right-handed rule (e.g. axis 1 cross axis 2 = axis 3).
                        </documentation>
                    </annotation>
                    <complexType>
                        <simpleContent>
                            <extension base="string">
                                <attribute name="name" type="NCName" use="required"/>
                            </extension>
                        </simpleContent>
                    </complexType>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="SpatialFramePropertyType">
    <sequence minOccurs="0">
        <element ref="sml:SpatialFrame"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

The *SpatialFrame* element provides a textual description of the physical process' local coordinate reference frame. The position of this physical component or system is based on relating this local reference frame to some external reference frame (e.g. a platform or geographic reference frame).

NOTE WELL: It is important that the following assumptions are understood when defining a local spatial reference frame:

(a) there is an assumption of a right-handed coordinate system (i.e. x cross y = z)
(b) the listing of the axes is assumed to be in order according to the right-hand rule (i.e. axis1 cross axis2 = axis3)

Failure to adher to these assumptions cannot be determined by a conformance test but can have dire consequences when tasking an actuator or geolocating sensor data. Failure to follow these assumptions typically results in negation of one or more axes or incorrect rotation directions about one or more axes.

An example of a local reference frame description is given below:

```
<sml:localReferenceFrame>
    <sml:SpatialFrame id="SENSOR_FRAME">
      <sml:origin>
            Origin is at the intersection of the symmetric axis of the cylinder and the rear of the sensor
      </sml:origin>
      <sml:axis name="X">X is perdendicular to the symmetric axis and intersects the indicator marked "x"
            on the casing</sml:axis>
      <sml:axis name="Y">Y is orthogonal to Z and X (= Z cross X)</sml:axis>
      <sml:axis name="Z">Z is along the symmetric axis of the sensor in the direction of view</sml:axis>
    </sml:SpatialFrame>
 </sml:localReferenceFrame>
```

8.4.1.3 Position

The *position* property of a physical component can be defined by one of several methods, including (1) by description, (2) by point, (3) by position, (4) by dynamic state, (5) by trajectory, or (6) by process. These are defined by the *PositionUnion* element shown below:

```
    <group name="PositionUnion">
        <annotation>
            <documentation>
                A choice of elements for defining spatial state (e.g. location, orientation, linear/angular velocity and
                linear/angular acceleration.
            </documentation>
        </annotation>
        <choice>
            <element ref="swe:Text">
                <annotation>
                    <documentation>
                        Provides positional information in textual form (e.g. "located on the intake line before the catalytic
                        converter"); shall only be used when a more precise location is unknown or irrelevant.
                    </documentation>
                </annotation>
            </element>
            <element ref="gml:Point">
                <annotation>
```

```
                <documentation>Provides static location only using a gml:Point element.</documentation>
            </annotation>
        </element>
        <element ref="swe:Vector">
            <annotation>
                <documentation>Provides a static location using a swe:Vector.</documentation>
            </annotation>
        </element>
        <element ref="swe:DataRecord">
            <annotation>
                <documentation>
                    Provides location and orientation as a DataRecord consisting of one or two Vector elements.
                </documentation>
            </annotation>
        </element>
        <element ref="swe:DataArray">
            <annotation>
                <documentation>
                    Provides time-tagged dynamic state information that can include, for instance, location,
                    orientation, velocity, acceleration, angular velocity, angular acceleration; shall be a DataArray
                    consisting of a DataRecord element of multiple Vector fields.
                </documentation>
            </annotation>
        </element>
        <element ref="sml:AbstractProcess">
            <annotation>
                <documentation>
                    Provides for positional information to be provided by a process; example processes could include
                    a physical sensor such as a GPS, a computational process such as an orbital propagation model,
                    a specific web service such as a SOS, or any process who's output provides positional
                    information.
                </documentation>
            </annotation>
        </element>
    </choice>
</group>

<complexType name="PositionUnionPropertyType">
    <sequence minOccurs="0">
        <group ref="sml:PositionUnion"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

The location of a component is the location of origin of the components local reference frame relative to the origin of some defined external reference frame (e.g. the reference frame of a sensor's platform or a geospatial reference frame). Likewise, the orientation of the component is the angular displacement of the x-y-z axes of the local reference frame relative to the corresponding axes of the external reference frame. For dynamic state, the velocity and acceleration are the first and second time derivatives of location, respectively, while angular velocity and angular acceleration are the first and second time derivatives of orientation.

Each option is defined in more detail below with appropriate examples.

8.4.1.3.1 Position by Description

There are physical components for which the precise location is either not known or is not really relevant. One example would be an O2 sensor on the exhaust line of a car as shown in the following example:

```
<sml:position>
```

```
    <swe:Text>Located on the exhaust manifold before the catalytic converter</swe:Text>
  </sml:position>
```

8.4.1.3.2 Position by Point

Position by point supports a static location using the *gml:Point* element. It is not appropriate if the orientation of the component is relevant or the component is has a dynamic location.

```
<sml:position>
  <!-- EPSG 4326 is for latitude-longitude, in that order -->
  <gml:Point gml:id="stationLocation" srsName="http://www.opengis.net/def/crs/EPSG/0/4326">
    <gml:coordinates>47.8 88.56</gml:coordinates>
  </gml:Point>
</sml:position>
```

8.4.1.3.3 Position by Location

As an alternative to using the *gml:Point*, a static location can also be provided using a *swe:Vector*, as in the example below:

```
<sml:position>
  <swe:Vector definition="http://sensorml.com/ont/swe/property/SensorLocation"
              referenceFrame="http://www.opengis.net/def/crs/EPSG/6.7/4326">
    <swe:coordinate name="Lat">
      <swe:Quantity definition="http://sensorml.com/ont/swe/property/Latitude" axisID="Lat">
        <swe:uom code="deg"/>
        <swe:value>47.8</swe:value>
      </swe:Quantity>
    </swe:coordinate>
    <swe:coordinate name="Lon">
      <swe:Quantity definition="http://sensorml.com/ont/swe/property/Longitude" axisID="Long">
        <swe:uom code="deg"/>
        <swe:value>2.3</swe:value>
      </swe:Quantity>
    </swe:coordinate>
  </swe:Vector>
</sml:position>
```

8.4.1.3.4 Position by Position

Position is defined here as consisting of both location and orientation. A static position should be provided using a *swe:DataRecord* consisting of two *swe:Vector* elements, one for location and one for orientation.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/position-by-position

Req 75. The *position* property shall take a *swe:DataRecord* as its value when both location and orientation are relevant. This *swe:DataRecord* shall consist of two *swe:Vector* fields, the first for location and the second for orientation.

An example of location and orientation defined for the position is shown below:

```
<sml:position>
  <swe:DataRecord>
    <swe:field name="location">
      <swe:Vector
           definition="http://sensorml.com/ont/swe/property/SensorLocation"
           referenceFrame="http://www.opengis.net/def/crs/EPSG/6.7/4979"
           localFrame="#SENSOR_FRAME">
        <swe:coordinate name="Lat">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/Latitude" axisID="Lat">
            <swe:uom code="deg"/>
            <swe:value>47.8</swe:value>
          </swe:Quantity>
        </swe:coordinate>
        <swe:coordinate name="Lon">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/Longitude" axisID="Long">
            <swe:uom code="deg"/>
            <swe:value>2.3</swe:value>
          </swe:Quantity>
        </swe:coordinate>
        <swe:coordinate name="Alt">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/Altitude" axisID="Alt">
            <swe:uom code="m"/>
            <swe:value>40.8</swe:value>
          </swe:Quantity>
        </swe:coordinate>
      </swe:Vector>
    </swe:field>
    <swe:field name="orientation">
      <swe:Vector
           definition="http://sensorml.com/def/property/SensorOrientation"
           referenceFrame="http://www.opengis.net/def/crs/NED"
           localFrame="#SENSOR_FRAME">
        <swe:coordinate name="TrueHeading">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/TrueHeading" axisID="Z">
            <swe:uom code="deg"/>
            <swe:value>-6.8</swe:value>
          </swe:Quantity>
        </swe:coordinate>
        <swe:coordinate name="Pitch">
          <swe:Quantity definition="http://sensorml.com/ont/swe/property/Pitch" axisID="Y">
            <swe:uom code="deg"/>
            <swe:value>0.3</swe:value>
          </swe:Quantity>
        </swe:coordinate>
      </swe:Vector>
    </swe:field>
  </swe:DataRecord>
</sml:position>
```

8.4.1.3.5 Position by Trajectory

For components moving within an external reference frame, the state of the component through time should be expressed using a *swe:DataArray* or process (described in the following section).

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/dynamic-state
Req 76. The dynamic state through time of a moving physical component shall be described using a *swe:DataArray* or a *sml:AbstractProcess*.

The data array should describe the time-dependent state of the physical component as appropriate and may include location, orientation, linear and angular velocity, and linear and angular acceleration.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/position-by-trajectory
Req 77. A *swe:DataArray* describing the trajectory of a moving component shall contain as its *elementType* a *swe:DataRecord* that includes a time field plus one or more *swe:Vector* fields supporting any or all appropriate properties of dynamic state (e.g. location, orientation, linear velocity, linear acceleration, angular velocity, and angular acceleration).

An example of dynamic position being defined as a trajectory is shown below. In this example the values for the location are provided inline within the *DataArray* element.

```
<sml:position>
  <swe:DataArray definition="http://www.opengis.net/def/trajectory">
    <swe:elementCount>
      <swe:Count>
        <swe:value>10</swe:value>
      </swe:Count>
    </swe:elementCount>
    <swe:elementType name="trajectory">
      <swe:DataRecord definition="http://sensorml.com/ont/swe/property/Trajectory">
        <swe:field name="samplingTime">
          <swe:Time definition="http://sensorml.com/ont/swe/property/SamplingTime">
            <swe:label>Sampling Time</swe:label>
            <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
          </swe:Time>
        </swe:field>
        <swe:field name="location">
          <swe:Vector
           definition="http://sensorml.com/ont/swe/property/Location"
           referenceFrame="http://www.opengis.net/def/crs/EPSG/6.7/4326"
           localFrame="#SENSOR_CRS">
           <swe:label>Platform Location</swe:label>
           <swe:coordinate name="Lat">
             <swe:Quantity definition="http://sensorml.com/ont/swe/property/Latitude"
```

```
                axisID="Lat">
                  <swe:uom code="deg"/>
                </swe:Quantity>
              </swe:coordinate>
              <swe:coordinate name="Lon">
                <swe:Quantity definition="http://sensorml.com/ont/swe/property/Longitude"
                  axisID="Long">
                  <swe:uom code="deg"/>
                </swe:Quantity>
              </swe:coordinate>
            </swe:Vector>
          </swe:field>
        </swe:DataRecord>
      </swe:elementType>
      <swe:encoding>
        <swe:TextEncoding blockSeparator="&#10;" tokenSeparator=","/>
      </swe:encoding>
      <swe:values>
        2011-03-01T04:20:00Z,25.72,-61.75
        2011-03-14T13:10:00Z,25.49,-61.70
        2011-03-21T18:43:00Z,25.35,-61.63
        2011-03-30T05:13:00Z,24.87,-61.43
        2011-04-08T01:45:00Z,24.86,-61.42
        2011-04-12T08:34:00Z,24.32,-61.67
        2011-04-15T09:12:00Z,24.54,-61.53
        2011-04-21T03:21:00Z,24.53,-61.68
        2011-04-27T04:34:00Z,24.32,-61.76
        2011-05-01T12:01:00Z,24.28,-61.56
      </swe:values>
    </swe:DataArray>
  </sml:position>
```

It is also possible to support "out-of-band" values by referencing an external "flat file" through the *xlink:href* attribute of the *values* property, such as:

```
    <swe:values xlink:href="http://myDomain/data/aircraftLocation_20048763">
```

Futhermore, the trajectory does not need to be constrained to provide only location. A trajectory *DataRecord* could include time, location, and orientation (e.g. pitch, roll, yaw), as well as perhaps velocity, and acceleration.
Finally, if part of the physical process (e.g. a sensor system) outputs its own position (e.g. using a GPS), that DataRecord can be referenced as in the following example:

```
<sml:outputs>
  <sml:OutputList>
    <!-- for the case of moving thermometer output in sync with GPS location -->
    <sml:output name="tempStream">
      <swe:DataRecord>
        <swe:field name="sampleTime">
          <swe:Time definition="http://sensorml.com/ont/swe/property/SamplingTime"
            referenceFrame="http://www.opengis.net/def/trs/OGC/0/GPS">
            <swe:label>Sampling Time</swe:label>
            <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
          </swe:Time>
        </swe:field>
        <swe:field name="temperature">
          <swe:Quantity definition="http://sweet.jpl.nasa.gov/2.2/quanTemperature.owl#Temperature">
            <swe:label>Air Temperature</swe:label>
```

```
            <swe:uom code="Cel"/>
          </swe:Quantity>
        </swe:field>
        <swe:field name="location" >
          <swe:Vector id="MY_LOCATION" definition="http://sensorml.com/ont/swe/property/Location"
                referenceFrame="http://www.opengis.net/def/crs/EPSG/6.7/4326">
            <swe:coordinate name="Lat">
              <swe:Quantity definition="http://sensorml.com/ont/swe/property/Latitude" axisID="Lat">
                <swe:uom code="deg"/>
              </swe:Quantity>
            </swe:coordinate>
            <swe:coordinate name="Lon">
              <swe:Quantity definition="http://sensorml.com/ont/swe/property/Longitude" axisID="Long">
                <swe:uom code="deg"/>
              </swe:Quantity>
            </swe:coordinate>
          </swe:Vector>
        </swe:field>
      </swe:DataRecord>
    </sml:output>
  </sml:OutputList>
</sml:outputs>
<!-- ================================== -->
<!--   Sensor Location from Component Output    -->
<!-- ================================== -->
<sml:position  xlink:href="#MY_LOCATION"/>
```

8.4.1.3.6 Position by Process

With dynamic sensor systems, it is often necessary or more efficient to calculate position or dynamic state as-needed, on-demand or to retrieve position values from a web service as-needed.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/position-by-process
Req 78. The position property shall take a SensorML-encoded process (either inline or by reference) when those values are calculated or retrieved on-demand as-needed. The output of that process shall be a *swe:DataArray* describing the trajectory of the component, or a *swe:DataRecord* describing instantaneous position or state of the component.

The following example shows a configured SimpleProcess that calculates position on-demand using an orbital mechanics model. The process takes time as input and provides dynamic state as output. The example utilizes a configurable process to be discussed in a later section.

```
<sml:position>
  <sml:SimpleProcess gml:id="OrbitalPropagationModel">
    <sml:validTime>
      <gml:TimePeriod gml:id="TimeSegment">
        <gml:beginPosition>2010-03-01T00:20:00Z</gml:beginPosition>
        <gml:endPosition>2010-03-02T04:01:00Z</gml:endPosition>
      </gml:TimePeriod>
    </sml:validTime>
```

```
    <sml:typeOf xlink:title="urn:sensors:sgp4" xlink:href="http://myDomain.org/processes/sgp4Propagation.xml"/>
     <sml:configuration>
       <sml:Settings>
         <sml:setValue ref="base/parameters/elements/epoch">2010-03-01T02:00:14.32994Z</sml:setValue>
          <sml:setValue ref="base/parameters/elements/star">0.11897E-4</sml:setValue>
         <sml:setValue ref="base/parameters/elements/inclination">98.7187</sml:setValue>
         <sml:setValue ref="base/parameters/elements/rightAscension">128.3968</sml:setValue>
         <sml:setValue ref="base/parameters/elements/eccentricity">0.00000571</sml:setValue>
        <sml:setValue ref="base/parameters/elements/argOfPerigee">101.8476</sml:setValue>
         <sml:setValue ref="base/parameters/elements/meanAnomaly">258.2808</sml:setValue>
         <sml:setValue ref="base/parameters/elements/meanMotion">14.20027191</sml:setValue>
       </sml:Settings>
     </sml:configuration>
     <sml:method xlink:href="http://blah.blah/processes/sgp4_method.xml"/>
   </sml:SimpleProcess>
 </sml:position>
```

The following example shows position obtained as-needed from an oline SOS web service. The SimpleProcess is a configured instance of an SOS client process. The process takes time as input and provides position as output.

```
<sml:position>
  <!-- position by process -->
  <sml:SimpleProcess gml:id="SOS_Client">
    <sml:typeOf xlink:title="urn:ogc:service:sos"
      xlink:href="http://blah.blah/def/processes/SOS-position-client.xml"/>
    <sml:configuration>
      <sml:Settings>
        <sml:setValue ref="base/parameters/URI">http://sensors.ws/services/mySOS</sml:setValue>
        <sml:setValue ref="base/parameters/layer">position</sml:setValue>
        <sml:setValue ref="base/parameters/procedure">mySat1</sml:setValue>
      </sml:Settings>
    </sml:configuration>
    <sml:method xlink:href="http://blah.blah/processes/position_sos_method.xml"/>
  </sml:SimpleProcess>
</sml:position>
```

8.4.1.4 Local Time Frame

The *localTimeFrame* property takes a *TemporalFrame* as its value and is an XML schema implementation of the *localTimeFrame* UML model defined in clause 7.6. The schema snippet for this element and its corresponding complex type is shown below.

```
<element name="TemporalFrame" type="sml:TemporalFrameType"
    substitutionGroup="swe:AbstractSWEIdentifiable">
    <annotation>
        <documentation>
            A general temporal frame such as a mission start time or timer start time. The origin should just describe
            context of the start of time (e.g. start of local timer).
        </documentation>
    </annotation>
</element>

<complexType name="TemporalFrameType">
    <complexContent>
        <extension base="swe:AbstractSWEIdentifiableType">
            <sequence>
                <element name="origin" type="string"/>
            </sequence>
```

```
            </extension>
        </complexContent>
    </complexType>

    <complexType name="TemporalFramePropertyType">
        <sequence minOccurs="0">
            <element ref="sml:TemporalFrame"/>
        </sequence>
        <attributeGroup ref="swe:AssociationAttributeGroup"/>
    </complexType>
```

An example of a declared localTimeFrame is given below:

```
<sml:localTimeFrame>
    <sml:TemporalFrame id="MISSION-START-TIME">
        <sml:origin>
            ISO 8601 formatted time at the start of the mission (i.e. when onboard clock is started)
        </sml:origin>
    </sml:TemporalFrame>
</sml:localTimeFrame>
```

8.4.1.5 Time Position

The *timePosition* property takes a *swe:TimePropertyType* element as its value. The *timePosition* property allows one to specify the relationship between a local time frame and some other internal or external time frame. as described above (Section 8.4.1.4).

The *id* of the *localTimeFrame* defined in the example above, can be used as the value of the *localFrame* attribute of the *swe:Time* element in order to specify the relationship of that local time frame to some external time frame. This way several time positions can be defined relative to each other. The following example shows how this can be used to express times of high frequency scan lines acquired by an airborne scanner relative to the flight's start time:

```
<sml:timePosition name="mission-start-time">
    <swe:Time definition="http://sensorml.com/ont/swe/property/MissionStartTime"
        localFrame="#MISSION-START-TIME" referenceFrame="http://www.opengis.net/def/trs/OGC/0/UTC">
        <swe:label>Flight Time</swe:label>
        <swe:description>Time at take-off in UTC</swe:description>
        <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
        <swe:value>2009-01-26T10:21:45+01:00</swe:value>
 </swe:Time>
</sml:timePosition>
```

Scan times can then be expressed relative to the flight's start time as in the example below (note: the scan-start-time position might be defined in the SensorML document, as below, but the high-frequency scan-start-time values would more likely be provided as part of a block or stream of output data).

```
<sml:timePosition name=”scan-start-time”>
    <swe:Time definition="http://sensorml.com/ont/swe/property/ScanStartTime"
              localFrame="#SCAN-START-TIME" referenceFrame="#MISSION-START-TIME">
        <swe:label>Scanline Time</swe:label>
        <swe:description>Acquisition time of the scan line</swe:description>
        <swe:uom code="s"/>
        <swe:value>1256.235</swe:value>
  </swe:Time>
</sml:timePosition>
```

8.4.2 Physical Component

PhysicalComponent is derived from *AbstractPhysicalProcess* and is the XML schema implementation of the “*PhysicalComponent*” UML class defined in clause 7.6. The schema snippet for this element and its corresponding complex type is shown below.

```
<element name="PhysicalComponent" type="sml:PhysicalComponentType"
    substitutionGroup="sml:AbstractPhysicalProcess">
    <annotation>
        <documentation>
            A PhysicalComponent is a physical process that will not be further divided into smaller components.
        </documentation>
    </annotation>
</element>
<complexType name="PhysicalComponentType">
    <complexContent>
        <extension base="sml:AbstractPhysicalProcessType">
            <sequence>
                <element name="method" type="sml:ProcessMethodPropertyType" minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            The method describes (as an algorithm or text) how the process takes the input and,
                            based on the parameter values, generates output values.
                        </documentation>
                    </annotation>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="PhysicalComponentPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:PhysicalComponent"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

The example below illustrates the definition of a local coordinate reference frame as well as the definition of position (i.e. both location and orientation) of the physical component. While the position of the component is specified here relative to a geospatial reference frame, it could also be specified relative to some other external reference frame, such as the reference frame of a platform.

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:PhysicalComponent gml:id="MY_SENSOR"
    xmlns:sml="http://www.opengis.net/sensorML/2.0"
    xmlns:swe="http://www.opengis.net/swe/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xsi:schemaLocation="http://www.opengis.net/sensorML/2.0 http://schemas.opengis.net/sensorml/2.0/sensorML.xsd">

  <!-- ================================================ -->
  <!--              System Description                  -->
  <!-- ================================================ -->
  <gml:description>
    Stationary Location and Orientation - Single-beam Motion Detector
  </gml:description>

  <!-- ================================================ -->
  <!--           Observed Property = Output             -->
  <!-- ================================================ -->
  <sml:outputs>
    <sml:OutputList>
      <sml:output name="motionDetection">
        <swe:Boolean definition="http://sensorml.com/ont/swe/property/Motion"/>
      </sml:output>
    </sml:OutputList>
  </sml:outputs>

  <!-- ================================== -->
  <!--     Station Reference Frame        -->
  <!-- ================================== -->
  <!-- Particularly when dealing with orientations, one needs to understand the sensor's
       reference frame as defined by the OEM or sensor deployer; the position and
       orientation of this local frame is then described relative to an external reference
       frame in the "position" property -->
  <sml:localReferenceFrame>
    <sml:SpatialFrame id="SENSOR_FRAME">
      <sml:origin>
        Origin is at the intersection of the symmetric axis of the cylinder and the rear of the sensor
      </sml:origin>
      <sml:axis name="X">
        X is perdendicular to the symmetric axis and intersects the indicator marked "x" on the casing
      </sml:axis>
      <sml:axis name="Y">Y = Z cross X</sml:axis>
      <sml:axis name="Z">
        Z is along the symmetric axis of the sensor in the direction of view
      </sml:axis>
    </sml:SpatialFrame>
  </sml:localReferenceFrame>

  <!-- ================================================ -->
  <!--     Station Location and Orientation             -->
  <!-- ================================================ -->
  <sml:position>
    <swe:DataRecord>
      <swe:field name="location">
        <swe:Vector
            definition="http://sensorml.com/ont/swe/property/SensorLocation"
            referenceFrame="http://www.opengis.net/def/crs/EPSG/6.7/4979"
            localFrame="#SENSOR_FRAME">
          <swe:coordinate name="Lat">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Latitude" axisID="Lat">
              <swe:uom code="deg"/>
              <swe:value>47.8</swe:value>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="Lon">
```

```
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Longitude" axisID="Long">
              <swe:uom code="deg"/>
              <swe:value>2.3</swe:value>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="Alt">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Altitude" axisID="Alt">
              <swe:uom code="m"/>
              <swe:value>40.8</swe:value>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:field>
      <swe:field name="orientation">
        <swe:Vector
            definition="http://sensorml.com/def/property0/SensorOrientation"
            referenceFrame="http://www.opengis.net/def/crs/NED"
            localFrame="#SENSOR_FRAME">
          <swe:coordinate name="TrueHeading">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/TrueHeading" axisID="Z">
              <swe:uom code="deg"/>
              <swe:value>-6.8</swe:value>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="Pitch">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/Pitch" axisID="Y">
              <swe:uom code="deg"/>
              <swe:value>0.3</swe:value>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:field>
    </swe:DataRecord>
  </sml:position>
</sml:PhysicalComponent>
```

8.5 Requirements Class: Physical System Schema

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-system	
Target Type	XML Instance
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component

XML Schema elements and types defined in the "*physical_system.xsd*" schema file implement all classes defined respectively in the "physical-system" UML package defined in Clause 7.7.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-system/schema-valid
Req 79. The XML instance shall be valid with respect to the XML grammar defined in the "physical_system.xsd", as well as satisfy all Schematron patterns defined in "physical_system.sch".

The following XML snippet provides the appropriate header and import statements for the physical_component.xsd:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:sml="http://www.opengis.net/sensorML/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:swe="http://www.opengis.net/swe/2.0"
    targetNamespace="http://www.opengis.net/sensorML/2.0" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <annotation>
        <documentation>
            The elements of a concrete aggregate physical process derived from the core process model.
        </documentation>
    </annotation>
    <import namespace="http://www.opengis.net/swe/2.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/2.0/swe.xsd"/>
    <import namespace="http://www.opengis.net/gml/3.2"
        schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
    <include schemaLocation="physical_component.xsd"/>
    <include schemaLocation="aggregate_process.xsd"/>
```

8.5.1 Physical System

The *PhysicalSystem* element and complex type are derived from the *AbstractPhysicalProcessType* and ths has all of the inherited properties defined in Clause

7.6. *PhysicalSystem* also has the components and connections properties that have already been described in Clause 7.4.
The schema snippet for this element and its corresponding complex type is shown below.

```
<element name="PhysicalSystem" type="sml:PhysicalSystemType"
    substitutionGroup="sml:AbstractPhysicalProcess">
    <annotation>
        <documentation>
            A PhysicalSystem is an aggregate system that can include multiple components (both physical and non-
            physical) with explicit links between the outputs, inputs, and parameters of the individual components. In a
            PhysicalSystem, the spatial position of the System itself is relevant to its application.
        </documentation>
    </annotation>
</element>
<complexType name="PhysicalSystemType">
    <complexContent>
        <extension base="sml:AbstractPhysicalProcessType">
            <sequence>
                <element name="components" type="sml:ComponentListPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            The collection of processes that make up a process aggregation.
                        </documentation>
                    </annotation>
                </element>
                <element name="connections" type="sml:ConnectionListPropertyType"
                    minOccurs="0" maxOccurs="1">
                    <annotation>
                        <documentation>
                            The explicit definition of data links between outputs, inputs, and parameters of the
                            components within an aggregate process.
                        </documentation>
                    </annotation>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="PhysicalSystemPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:PhysicalSystem"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

A *PhysicalSystem* is a physical process that consists of multiple components that can be both physical and non-physical. As with the non-physical *AggregateProcess*, the *PhysicalSystem* allows for listing of the components as well as explicit mapping of the flow of data throughout the system.
In the example below there are three *components*: a thermometer that outputs temperature, an anemometer that outputs both wind speed and wind direction, and a non-physical windchill process that takes temperature and wind speed and calculates a windchill factor. The components are provided “by reference” using the *xlink:href* attribute. The outputs of these three components connect to the outputs of the system itself, as shown in the *ConnectionsList*.

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:PhysicalSystem gml:id="MY_WEATHER_STATION"
  xmlns:sml="http://www.opengis.net/sensorML/2.0"
  xmlns:swe="http://www.opengis.net/swe/2.0"
  xmlns:gml="http://www.opengis.net/gml/3.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xsi:schemaLocation="http://www.opengis.net/sensorML/2.0 http://schemas.opengis.net/sensorml/2.0/sensorML.xsd">

  <!-- ================================================ -->
  <!--               System Description                 -->
  <!-- ================================================ -->
  <gml:description> Weather station in my yard </gml:description>
  <gml:identifier codeSpace="uid">urn:weather-we-is:stations:FR8766</gml:identifier>

  <!-- ================================================ -->
  <!--            Inputs  = Observed Properties         -->
  <!-- ================================================ -->
    <sml:inputs>
        <sml:InputList>
            <sml:input name="temperature">
                <sml:ObservableProperty definition="http://sweet.jpl.nasa.gov/2.3/propTemperature.owl#Temperature"/>
            </sml:input>
            <sml:input name="wind">
                <sml:ObservableProperty definition="http://sweet.jpl.nasa.gov/2.3/phenAtmoWind.owl#Wind"/>
            </sml:input>
        </sml:InputList>
    </sml:inputs>

  <!-- ================================================ -->
  <!--           Outputs = Quantities                   -->
  <!-- ================================================ -->
  <sml:outputs>
    <sml:OutputList>
      <sml:output name="weather">
        <swe:DataRecord>
          <swe:field name="temperature">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/AtmosphericTemperature">
              <swe:label>Air Temperature</swe:label>
              <swe:uom code="cel"/>
            </swe:Quantity>
          </swe:field>
          <swe:field name="wind_chill">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/WindChillFactor">
              <swe:label>Wind Chill Factor</swe:label>
              <swe:uom code="cel"/>
            </swe:Quantity>
          </swe:field>
          <swe:field name="wind_speed">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/WindSpeed">
              <swe:label>Wind Speed</swe:label>
              <swe:uom code="km/h"/>
            </swe:Quantity>
          </swe:field>
          <swe:field name="wind_direction">
            <swe:Quantity definition="http://sensorml.com/ont/swe/property/WindDirection">
              <swe:label>Wind Direction</swe:label>
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </sml:output>
    </sml:OutputList>
  </sml:outputs>
```

```
<!-- =============================================== -->
<!--               System Location                   -->
<!-- =============================================== -->
<sml:position>
  <gml:Point gml:id="stationLocation" srsName="http://www.opengis.net/def/crs/EPSG/0/4326">
    <gml:coordinates>47.8 88.56</gml:coordinates>
  </gml:Point>
</sml:position>

<!-- =============================================== -->
<!--               System Components                 -->
<!-- =============================================== -->
<sml:components>
  <sml:ComponentList>
    <sml:component name="thermometer" xlink:title="urn:davis:sensors:7817"
      xlink:href="http://www.sensors.ws/examples/SensorML-2.0/xml/sensors/Davis_7817_complete.xml"/>
    <sml:component name="anemometer" xlink:title="urn:davis:sensors:barometer_internal"
      xlink:href="http://www.sensors.ws/examples/SensorML-2.0/xml/sensors/Davis_7911.xml"/>
     <sml:component name="windchill" xlink:title="urn:ogc:process:windchill-02"
      xlink:href="http://www.sensors.ws/examples/SensorML-2.0/xml/processes/windchill-02.xml"/>
  </sml:ComponentList>
</sml:components>

<!-- =============================================== -->
<!--   Connections between components and system output  -->
<!-- =============================================== -->
<sml:connections>
  <sml:ConnectionList>
    <!-- connection between thermometer's output and system's temperature output -->
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/thermometer/outputs/temperature"/>
        <sml:destination ref="outputs/weather/temperature"/>
      </sml:Link>
    </sml:connection>
    <!-- connection between anemometer's wind speed output and system's windspeed output -->
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/anemometer/outputs/wind_state/wind_speed"/>
        <sml:destination ref="outputs/weather/wind_speed"/>
      </sml:Link>
    </sml:connection>
    <!-- connection between anemometer's wind direction output and system's wind direction output -->
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/anemometer/outputs/wind_state/wind_direction"/>
        <sml:destination ref="outputs/weather/wind_direction"/>
      </sml:Link>
    </sml:connection>
    <!-- connection between thermometer's output and windchill temperature input  -->
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/thermometer/outputs/temperature"/>
        <sml:destination ref="components/windchill/inputs/process_inputs/temperature"/>
      </sml:Link>
    </sml:connection>
    <!-- connection between anemometer's wind speed output and windchill wind_speed input -->
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/thermometer/outputs/wind_state/wind_speed"/>
        <sml:destination ref="components/windchill/inputs/process_inputs/wind_speed"/>
      </sml:Link>
    </sml:connection>
    <!-- connection between windchill process output and system's windchill output -->
    <sml:connection>
      <sml:Link>
```

```
            <sml:source ref="components/thermometer/outputs/windchill"/>
            <sml:destination ref="outputs/weather/windchill"/>
          </sml:Link>
        </sml:connection>
      </sml:ConnectionList>
    </sml:connections>
</sml:PhysicalSystem>
```

Requirements Class: Configurable Process Schema

Requirements Class	
http://www.opengis.net/spec/sensorml/2.0/req/xml/configuration	
Target Type	XML Instance
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/model/configuration
Dependency	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process

XML Schema elements and types defined in the "*configurable_process.xsd*" schema file implement all classes defined respectively in the "configurable_process" UML package defined in Clause 7.7.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/configuration/schema-valid
Req 80. The XML instance shall be valid with respect to the XML grammar defined in the "configuration.xsd", as well as satisfy all Schematron patterns defined in "configuration.sch".

The following XML snippet provides the appropriate header and import statements for the configurable_process.xsd:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:sml="http://www.opengis.net/sensorML/2.0"
    xmlns:swe="http://www.opengis.net/swe/2.0" targetNamespace="http://www.opengis.net/sensorML/2.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <annotation>
        <documentation>
            The elements supporting configurability and configuration settings of a instance of any process.
        </documentation>
    </annotation>
    <import namespace="http://www.opengis.net/swe/2.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/2.0/swe.xsd"/>
    <include schemaLocation="simple_process.xsd"/>
```

The primary elements supporting configurable processes will be discussed in more detail in the following subclauses.

8.5.2 Modes

The *Mode* element is an XML schema implementation of the models defined in Clause 7.9.1. It allows one to define a choice of modes, each of which in turn sets the values for a collection of parameters.
The XML snippet for the *Mode* and *ModeChoice* elements and their complex types is given below:

Mode:

```
<element name="Mode" type="sml:ModeType" substitutionGroup="sml:DescribedObject"/>

<complexType name="ModeType">
    <complexContent>
        <extension base="sml:DescribedObjectType">
            <sequence>
                <element name="configuration" type="sml:SettingsPropertyType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ModePropertyType">
    <sequence minOccurs="0">
        <element ref="sml:Mode"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

ModeChoice:

```
<element name="ModeChoice" type="sml:ModeChoiceType" substitutionGroup="sml:AbstractModes">
    <annotation>
        <documentation>
            A collection of modes from which one can exclusively select during configuration of a process.
        </documentation>
    </annotation>
</element>

<complexType name="ModeChoiceType">
    <complexContent>
        <extension base="sml:AbstractModesType">
            <sequence>
                <element name="mode" type="sml:ModePropertyType" minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="ModeChoicePropertyType">
    <sequence minOccurs="0">
        <element ref="sml:ModeChoice"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

A *Mode* is restricted in the values that it can set. In essence, a *Mode* is restricted to setting only the values of previously defined parameters of the enclosed process and its base processes, and the values of these parameters should be set consistent with the parameters allowed values.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/configuration/mode-restriction
Req 81. A *Mode* shall not set any process property values, other than the values of parameters defined within the same process or its parent processes.

Requirement
http://www.opengis.net/spec/sensorml/2.0/req/xml/configuration/parameter-values
Req 82. The parameter values set by a *Mode* cannot be outside of the values allowed by the *AllowedValues* property of the *parameter*.

In the following example, the definition of the parameters, samplingRate and gain are defined within the parameters section of the process description. Then two modes ("lowThreat" and "highThreat") are defined, each of which defines specific values for these the two parameters.

```
<!-- ================================== -->
<!--          Parameters                -->
<!-- ================================== -->
<sml:parameters>
   <sml:ParameterList>
          <sml:parameter name="settings">
              <swe:DataRecord id="CURRENT_SETTINGS">
                <swe:field name="samplingRate">
                      <swe:Quantity definition="http://sensorml.com/ont/swe/property/SamplingRate"
                          updatable="false">
                          <swe:label>Sampling Rate</swe:label>
                          <swe:uom code="Hz"/>
                          <swe:constraint>
                              <swe:AllowedValues>
                                  <swe:interval>0.01 10.0</swe:interval>
                              </swe:AllowedValues>
                          </swe:constraint>
                      </swe:Quantity>
                  </swe:field>
                  <swe:field name="gain">
                      <swe:Quantity definition="http://sensorml.com/ont/swe/property/Gain" updatable="false">
                          <swe:label>Gain</swe:label>
                          <swe:uom code="Hz"/>
                          <swe:constraint>
                              <swe:AllowedValues>
                                  <swe:interval>1.0 2.5</swe:interval>
                              </swe:AllowedValues>
                          </swe:constraint>
                      </swe:Quantity>
                  </swe:field>
                  <!-- Note: no parameter values provided because the sensor switches  automatically when mode is
                      changed -->
              </swe:DataRecord>
          </sml:parameter>
   </sml:ParameterList>
</sml:parameters>

<!-- ================================== -->
<!--              Modes                 -->
<!-- ================================== -->
<sml:modes>
   <sml:ModeChoice id="THREAT_LEVEL_MODE">
      <sml:mode>
         <sml:Mode gml:id="lowThreat">
            <gml:description> Setting when nothing has been detected </gml:description>
            <gml:name>Low Threat Mode</gml:name>
            <sml:configuration>
```

```
            <sml:Settings>
              <sml:setValue ref="parameters/settings/samplingRate">0.1</sml:setValue>
              <sml:setValue ref="parameters/settings/gain">1.0</sml:setValue>
            </sml:Settings>
          </sml:configuration>
        </sml:Mode>
      </sml:mode>
      <sml:mode>
        <sml:Mode gml:id="highThreat">
          <gml:description> Setting when something has been detected </gml:description>
          <gml:name>High Threat Mode</gml:name>
          <sml:configuration>
            <sml:Settings>
              <sml:setValue ref="parameters/settings/samplingRate">10.0</sml:setValue>
              <sml:setValue ref="parameters/settings/gain">2.5</sml:setValue>
            </sml:Settings>
          </sml:configuration>
        </sml:Mode>
      </sml:mode>
    </sml:ModeChoice>
  </sml:modes>
```

8.5.3 Settings

The *Settings* element is an XML schema implementation of the models defined in Clause 7.9.2. It allows one to set values of parameters, to select modes, and to enable or disable components.

The XML snippet for the *Settings* element and its complex types is given below:

```
<element name="Settings" type="sml:SettingsType" substitutionGroup="sml:AbstractSettings"/>

<complexType name="SettingsType">
    <complexContent>
        <extension base="sml:AbstractSettingsType">
            <sequence>
                <element name="setValue" type="sml:ValueSettingPropertyType"
                    minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>Allows the setting of a particular property value.</documentation>
                    </annotation>
                </element>
                <element name="setArrayValues" type="sml:ValueSettingPropertyType"
                    minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>Allows the setting of array values for a particular property.</documentation>
                    </annotation>
                </element>
                <element name="setConstraint" type="sml:ConstraintSettingPropertyType"
                    minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            Allows one to further restrain the allowed values of a particular property.
                        </documentation>
                    </annotation>
                </element>
                <element name="setMode" type="sml:ModeSettingPropertyType"
                    minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            Allows one to select a predefined mode, which by inference sets a collection of property
                            values according to the definition of that mode.
                        </documentation>
```

```
                    </annotation>
                </element>
                <element name="setStatus" type="sml:StatusSettingPropertyType"
                    minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>
                            Allows one to set the status (enabled, disabled, etc) of a particular input, output, or
                            parameter.
                        </documentation>
                    </annotation>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="SettingsPropertyType">
    <sequence minOccurs="0">
        <element ref="sml:Settings"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>
```

setValue PropertyType:

```
<complexType name="ValueSettingPropertyType">
    <simpleContent>
        <extension base="token">
            <attribute name="ref" type="sml:DataComponentPathPropertyType" use="required"/>
        </extension>
    </simpleContent>
</complexType>
```

setArrayValue PropertyType:

```
<complexType name="ArraySettingPropertyType">
    <sequence>
        <element name="ArrayValues">
            <complexType>
                <sequence>
                    <element name="encoding">
                        <complexType>
                            <sequence>
                                <element ref="swe:AbstractEncoding"/>
                            </sequence>
                        </complexType>
                    </element>
                    <element name="value" type="swe:EncodedValuesPropertyType"/>
                </sequence>
            </complexType>
        </element>
    </sequence>
    <attribute name="ref" type="sml:DataComponentPathPropertyType" use="required"/>
</complexType>
```

setMode PropertyType:

```
<complexType name="ModeSettingPropertyType">
    <simpleContent>
        <extension base="NCName">
            <attribute name="ref" type="sml:DataComponentPathPropertyType" use="required"/>
        </extension>
    </simpleContent>
</complexType>
```

setStatus PropertyType:

```
<complexType name="StatusSettingPropertyType">
    <simpleContent>
        <extension base="sml:StatusType">
            <attribute name="ref" type="sml:DataComponentPathPropertyType" use="required"/>
        </extension>
    </simpleContent>
</complexType>
```

setConstraint PropertyType:

```
<complexType name="ConstraintSettingPropertyType">
    <sequence>
        <group ref="sml:Constraint"/>
    </sequence>
    <attribute name="ref" type="sml:DataComponentPathPropertyType" use="required"/>
</complexType>

<group name="Constraint">
    <choice>
        <element ref="swe:AllowedTimes"/>
        <element ref="swe:AllowedTokens"/>
        <element ref="swe:AllowedValues"/>
    </choice>
</group>

<complexType name="ConstraintPropertyType">
    <sequence minOccurs="0">
        <group ref="sml:Constraint"/>
    </sequence>
    <attributeGroup ref="swe:AssociationAttributeGroup"/>
</complexType>

<simpleType name="StatusType">
    <restriction base="string">
        <enumeration value="enabled"/>
        <enumeration value="disabled"/>
    </restriction>
</simpleType>
```

The following example shows use of the *configuration* property within a configured process. In this example, the value of the Averaging Period for the raingauge is set to 60.0 (in whatever units of measure are use by the parameter), the Sampling Period of the raingauge has been constrained between 30.0 and 60.0, and the rain gauge component has been enabled.

The path describing the properties and components will be provided by the *ref* attribute and should follow the same rules set for connection *Links* in Section 8.3.3.2.

Annex A
(normative)

Abstract Conformance Test Suite for Models

A.1 Conformance Test Class: Core Concepts

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/core	
Target Type	Derived Encodings and Schema

Tests described in this section shall be used to test conformance of software and encoding models implementing the Requirements Class: Core Concepts (normative core).

A.1.1 Core concepts are the base of all derived models

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/core/core-concepts-used	
Requirement Req 1	http://www.opengis.net/spec/sensorml/2.0/req/core/core-concepts-used
Test Method	Inspect the schema or encoding definition to verify that it correctly implements the core model concepts.
Test Type	Capability

A.1.2 A process model has inputs, outputs, parameters, and method

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/core/processes	
Requirement Req 2	http://www.opengis.net/spec/sensorml/2.0/req/core/processes
Test Method	Inspect the schema or encoding definition to verify that the model defines inputs, outputs, parameters, and methodology.
Test Type	Capability

A.1.3 A process model has a unique ID

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/core/unique-id	
Requirement Req 3	http://www.opengis.net/spec/sensorml/2.0/req/core/unique-id
Test Method	Inspect the schema or encoding definition to verify that the process includes a unique ID.
Test Type	Capability

A.1.4 A process model has metadata

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/core/metadata	
Requirement Req 4	http://www.opengis.net/spec/sensorml/2.0/req/core/metadata
Test Method	Inspect the schema or encoding definition to verify that the core model includes metadata supporting identification, discovery, and qualification of the process.
Test Type	Capability

A.1.5 Metadata not used in process execution

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/core/execution	
Requirement Req 5	http://www.opengis.net/spec/sensorml/2.0/req/core/execution
Test Method	Verify that the implementation of the conceptual model has a constraint that all information required fro execution of a process is contained in the inputs, outputs, parameters, and methodology of the process.
Test Type	Capability

A.2 Conformance Test Class: Core Abstract Process

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process	
Target Type	Derived Encodings and Schema
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/core**
Dependency	**http://www.opengis.net/spec/SWE/2.0/conf/uml-block-components**
Dependency	ISO 19115:2006 (All Metadata)
Dependency	ISO 19136 (GML)

Tests described in this section shall be used to test conformance of software and encoding models implementing the conceptual models defined in Requirements Class: Core Abstract Process.

A.2.1 Dependency on Core

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/dependency-core	
Requirement Req 6	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/dependency-core
Test Method	Apply all tests in: http://www.opengis.net/spec/sensorml/2.0/conf/core
Test Type	Capability

A.2.2 Fully implement CoreProcess

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/package-fully-implement	
Requirement Req 7	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/package-fully-implement
Test Method	Inspect the schema or encoding definition to verify that it implements all classes in the “core-process” package.
Test Type	Capability

A.2.3 DescribedObject derived from GML AbstractFeature

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/gml-dependency	
Requirement Req 8	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/gml-dependency
Test Method	Inspect the schema or encoding definition to verify that all classes derived from *DescribedModel* are of type *featureType*.
Test Type	Capability

A.2.4 Using GML identifier for uniqueID in CoreProcess

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/unique-id	
Requirement Req 9	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/unique-id
Test Method	Inspect the schema or encoding definition to verify that it provides a unique identifier using the *gml:identifier* property.
Test Type	Capability

A.2.5 Extensions shall be in a separate namespace

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/extension-independence	
Requirement Req 10	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/extension-independence
Test Method	Inspect the schema or encoding definition to verify that any model used to define the value of the *extension* property exist within a separate namespace.
Test Type	Capability

A.2.6 Extensions shall not be required for process execution

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/extension-restrictions	
Requirement Req 11	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/extension-restrictions
Test Method	Inspect the schema or encoding definition to verify that information contained with the *extension* property is not required for execution of the process.
Test Type	Capability

A.2.7 ObservableProperty and SWE Common Data used for process input, output, and parameters

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/swe-common-dependency	
Requirement Req 12	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/swe-common-dependency
Test Method	Inspect the schema or encoding definition to verify that the value of inputs, outputs, and parameters are constrained to using SWE Common Data Block Components.
Test Type	Capability

A.2.8 Use of SWE Common Data aggregate models for process input, output, and parameters

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/aggregate-data	
Requirement Req 13	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/aggregate-data
Test Method	Inspect the schema or encoding definition to verify that tightly related data components are modeled within an appropriate SWE Common Data aggregate data model.
Test Type	Capability

A.2.9 Application and requirements of *typeOf* property

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/type-of	
Requirement Req 14	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/type-of
Test Method	Inspect the schema or encoding definition to verify that the *typeOf* property is a resolvable URL that references an instance of a process.
Test Type	Capability

A.2.10 Simple inheritance extends a base class referenced by typeOf

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/simple-inheritance	
Requirement Req 15	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/simple-inheritance
Test Method	Verify that the implementation of the conceptual model supports simple, additive inheritance through the use of the *typeOf* property.
Test Type	Capability

A.2.11 Supporting configuration in processes

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/configuration	
Requirement Req 16	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/configuration
Test Method	Inspect the schema or encoding definition to verify that model supports restriction of inherited properties through the *configuration* property.
Test Type	Capability

A.2.12 Dependency on SWE Common Data simple types

Conformance Test

http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process/swe-common-dependency	
Requirement Req 17	http://www.opengis.net/spec/sensorml/2.0/req/model/core-process/swe-common-dependency
Test Method	Validate that the encoding or schema pass the SWE Common Data conformance tests provided in “Records Components Package”.
Test Type	Capability

A.3 Conformance Test Class: Simple Process

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/model/simple-process	
Target Type	Derived Encodings and Schema
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process**

Tests described in this section shall be used to test conformance of software and encoding models implementing the conceptual models defined in Requirements Class: Simple Process.

A.3.1 Dependency on core

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/simple-process/dependency-core	
Requirement Req 18	http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/dependency-core
Test Method	Apply all tests in: http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process
Test Type	Capability

A.3.2 Fully Implement SimpleProcess

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/simple-process/package-fully-implement	
Requirement Req 19	http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/package-fully-implement
Test Method	Inspect the schema or encoding definition to verify that it implements all classes defined in “simple-process” package.
Test Type	Capability

A.3.3 Simple process definition

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/simple-process/definition	
Requirement Req 20	http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/definition
Test Method	Inspect the schema or encoding definition to verify that it supports well-defined inputs and outsputs but cannot be further divided into sub-processes.
Test Type	Capability

A.3.4 Simple process has method

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/simple-process/method	
Requirement Req 21	http://www.opengis.net/spec/sensorml/2.0/req/model/simple-process/method
Test Method	Inspect the schema or encoding definition to verify that it support definition of the process method.
Test Type	Capability

A.4 Conformance Test Class: Aggregate Process

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/model/aggregate-process	
Target Type	Derived Encodings and Schema
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process**

Tests described in this section shall be used to test conformance of software and encoding models implementing the conceptual models defined in Requirements Class: Aggregate Process.

A.4.1 Dependency on core

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/aggregate-process/dependency-core	
Requirement Req 22	http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/dependency-core
Test Method	Apply all tests in: http://www.opengis.net/spec/sensorml/2.0/conf/model/simple-process
Test Type	Capability

A.4.2 Fully Implement Aggregate Process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/aggregate-process/package-fully-implemented	
Requirement Req 23	http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/package-fully-implemented
Test Method	Inspect the schema or encoding definition to verify that it implements all classes within the “aggregate-process” package.
Test Type	Capability

A.4.3 Definition of Aggregate Process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/aggregate-process/definition	
Requirement Req 24	http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/definition
Test Method	Inspect the schema or encoding definition to verify that it supports well-defined inputs and outputs, and the ability to divide the process into sub-processes.
Test Type	Capability

A.4.4 Aggregate Process requires one or more components

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/aggregate-process/components	
Requirement Req 25	http://www.opengis.net/spec/sensorml/2.0/req/model/aggregate-process/components
Test Method	Inspect the schema or encoding definition to verify that the aggregate process has at least on sub-process.
Test Type	Capability

A.5 Conformance Test Class: Physical Component

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component	
Target Type	Derived Encodings and Schema
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process**

Tests described in this section shall be used to test conformance of software and encoding models implementing the conceptual models defined in Requirement Class: Physical Component.

A.5.1 Fully implement Physical Component

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component/package-fully-implement	
Requirement Req 26	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/package-fully-implement
Test Method	Inspect the schema or encoding definition to verify that it fully implements all classes of the "physical-component" package.
Test Type	Capability

A.5.2 Dependency on core process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component/dependency-core	
Requirement Req 27	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/dependency-core
Test Method	Apply all tests in: http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process
Test Type	Capability

A.5.3 Position by point

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component/by-point-or-location	
Requirement Req 28	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-point-or-location
Test Method	Inspect the schema or encoding to verify that when an object's position is specified "byPoint" or "byLocation" that the location is defined as a set of spatial coordinates relative to an external reference frame.
Test Type	Capability

A.5.4 Position by location and orientation

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component/by-position	
Requirement Req 29	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-position
Test Method	Inspect the schema or encoding definition to verify that, when an object's position is specified "byPosition", that the location of origin of the object's reference frame relative to an external frame is specified using one vector for location and that the object's orientation relative to an external frame is provided by another vector. Furthermore, verify that these vectors specify an external reference frame.
Test Type	Capability

A.5.5 Position by trajectory

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component/by-trajectory	
Requirement Req 30	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-trajectory
Test Method	Inspect the schema or encoding definition to verify that, when an object's position is provided "byTrajectory", that at least the location

	is provided as a time-tagged series of values.
Test Type	Capability

A.5.6 Position by process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component/by-process	
Requirement Req 31	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-process
Test Method	Inspect the schema or encoding definition to verify that, when an object's position is provided "byProcess" that position value is defined as a process modeled using SensorML, and that this process provides, at a minimum, a time-tagged series of location values reative to an external reference frame.
Test Type	Capability

A.5.7 Physical Component definition

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component/by-process	
Requirement Req 32	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-component/by-process
Test Method	Inspect the schema or encoding definition to verify that it includes well-defined inputs and outputs, that there is no ability to further divide the process into sub-processes, and that it provides a position property.
Test Type	Capability

A.6 Conformance Test Class: Physical System

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-system	
Target Type	Derived Encodings and Schema
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component**

Tests described in this section shall be used to test conformance of software and encoding models implementing the conceptual models defined in Requirements Class: Physical System.

A.6.1 Fully implement Physical System

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-system/package-fully-implemented	
Requirement Req 33	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system/package-fully-implemented
Test Method	Inspect the schema or encoding definition to verify that it implements all classes of the "physical-system" package.
Test Type	Capability

A.6.2 Physical System definition

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-system/definition	
Requirement Req 34	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system/definition
Test Method	Inspect the schema or encoding definition to verify that it includes well-defined inputs and outputs, that there is the ability to further divide the process into sub-processes, and that it provides a position property.
Test Type	Capability

A.6.3 Physical System dependency

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-system/dependency-core	
Requirement Req 35	http://www.opengis.net/spec/sensorml/2.0/req/model/physical-system/dependency-core
Test Method	Apply all test in: http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component
Test Type	Capability

A.7 Conformance Test Class: Process with Advanced Data Types

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/model/advanced-process	
Target Type	Derived Encodings and Schema
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process**
Dependency	**http://www.opengis.net/spec/SWE/2.0/conf/uml-block-components**
Dependency	**http://www.opengis.net/spec/SWE/2.0/conf/uml-choice-components**

Tests described in this section shall be used to test conformance of software and encoding models implementing the conceptual models define Requirements Class: Processes with Advanced Data Types.

A.7.1 Advanced Process dependency

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/advanced-process/dependency-core	
Requirement Req 36	http://www.opengis.net/spec/sensorml/2.0/req/model/advanced-process/dependency-core
Test Method	Apply all tests in: http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process
Test Type	Capability

A.7.2 Fully implement AdvancedProcess

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/advanced-process/package-fully-implement	
Requirement Req 37	http://www.opengis.net/spec/sensorml/2.0/req/model/advanced-process/package-fully-implement
Test Method	Inspect the schema or encoding definition to verify that it implements all classes within the “advanced-process” package.
Test Type	Capability

A.8 Conformance Test Class: Configurable Processes

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process	
Target Type	Derived Encoding and Software Implementation
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/conf/core-process**

Tests described in this section shall be used to test conformance of software and encoding models implementing the conceptual models defined in Requirements Class: Configurable Processes.

A.8.1 Dependency on Core Process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process/dependency-core	
Requirement Req 38	http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/dependency-core
Test Method	Apply all tests in: http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process
Test Type	Capability

A.8.2 Fully Implement Configurable Process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process/package-fully-implemented	
Requirement Req 39	http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/package-fully-implemented
Test Method	Inspect the schema or encoding definition to verify that it implements all classes within the “configuration” package.
Test Type	Capability

A.8.3 *ModeChoice* requires 2 or more *Modes*

Conformance Test

http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process/two-modes-required	
Requirement Req 40	http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/two-modes-required
Test Method	Inspect the schema or encoding definition to verify that it includes two or more *mode* properties.
Test Type	Capability

A.8.4 A configured process requires a Settings element

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process/settings-property	
Requirement Req 41	http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/settings-property
Test Method	Verify that the implementation of the configured process has a constraint that takes a *Settings* class as its value..
Test Type	Capability

A.8.5 Only *parameter* values can be set by *setValue*

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process/set-value-restriction	
Requirement Req 42	http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/set-value-restriction
Test Method	Verify that the *setValue* property of a configured process references only parameters defined within a configurable process.
Test Type	Capability

A.8.6 Only *parameter* array values can be set by *setArrayValues*

Conformance Test
http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process/set-array-value-

restriction	
Requirement Req 43	http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/set-array-value-restriction
Test Method	Verify that the *setArrayValue* property of a configured process references only array values for a parameter defined within a configurable process.
Test Type	Capability

A.8.7 Only *parameter* values can be constrained with *setConstraint*

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/model/configurable-process/set-constraint-restriction	
Requirement Req 44	http://www.opengis.net/spec/sensorml/2.0/req/model/configurable-process/set-constraint-restriction
Test Method	Verify that a *setConstraint* property of a configured process references a parameter defined within a configurable process.
Test Type	Capability

Annex B
(normative)

Abstract Conformance Test Suite for Schema

B.1 Conformance Test Class: Core Abstract Process Schema

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process	
Target Type	XML Instance
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/core-process**
Dependency	**http://www.opengis.net/spec/SWE/2.0/conf/xml-encoding-principles**
Dependency	**http://schemas.opengis.net/gml/3.2.1/gml.xsd**
Dependency	**http://schemas.opengis.net/sweCommon/2.0/swe.xsd**
Dependency	**http://schemas.opengis.net/iso/19139/20070417/gmd/gmd.xsd**
Dependency	**http://schemas.opengis.net/iso/19139/20070417/gco/gco.xsd**

All tests in this conformance test class and in the following shall be used to check conformance of XML instances created according to the schemas defined in this standard. They shall also be used to check conformance of software implementations that output XML instances.

B.1.1 Compliance with core XML schemas and Schematron patterns

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/schema-valid	
Requirement Req 45	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/schema-valid
Test Method	Validate the XML instance containing core process with the "core.xsd" XML schema file and the Schematron patterns in "core.sch".
Test Type	Capability

B.1.2 XML property values are included inline or by reference

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/ref-or-inline-value-present	
Requirement Req 46	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/ref-or-inline-value-present
Test Method	Check that all properties either include an inline value or an "xlink:href" attribute.
Test Type	Capability

B.1.3 Each extension uses a different namespace

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/extension-namespace-unique	
Requirement Req 47	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/extension-namespace-unique
Test Method	Test the value of the *extension* property to determine that the namespace of the root element is not http://schemas.opengis.net/sensorml/*.
Test Type	Capability

B.1.4 Extensions do not redefine XML elements or types

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/extension-coherent-with-core	
Requirement Req 48	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/extension-coherent-with-core
Test Method	Verify that all XML elements of the XML instance containing extensions can still be interpreted correctly without reading the extended information. *Note: This test cannot be run automatically as the meaning the extension shall be known and thus is not required to be implemented in the Executable Test Suite.*
Test Type	Capability

B.1.5 The value of the definition attribute is a resolvable URI

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/extension-process-execution	
Requirement Req 49	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/extension-process-execution
Test Method	Verify that the process execution does not require information to be retrieved from the extension element.
Test Type	Capability

B.1.6 Dependence on GML 3.2

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/gml-dependency	
Requirement Req 50	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/gml-dependency
Test Method	Validate the XML Instance according to GML 3.2 conformance tests
Test Type	Capability

B.1.7 Dependence on SWE Common Data 2.0

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/swe-common-dependency	
Requirement Req 51	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/swe-common-dependency
Test Method	Validate the XML Instance according to SWE Common Data v2.0 conformance tests
Test Type	Capability

B.1.8 Globally unique ID required

Conformance Test
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/globally-unique-id

Requirement Req 52	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/globally-unique-id
Test Method	Validate the XML Instance contains a unique ID for *gml:identifier* based on a well-defined protocol and that the value of the *codespace* attribute is "uniqueID".
Test Type	Capability

B.1.9 External namespace required for security constraints

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/document-security-tags	
Requirement Req 53	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/document-security-tags
Test Method	Validate that the XML Instance does not use the http://schemas.opengis.net/sensorml/* namespace for the value of the *securityConstraint* property.
Test Type	Capability

B.1.10 *Extension* element used for security tagging of individual properties

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/individual-security-tags	
Requirement Req 54	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/individual-security-tags
Test Method	Validate within a derived schema or XML Instance that security tagging of individual properties uses the *extension* element.
Test Type	Capability

B.1.11 Xlink role or arcrole shall be used to define relationship of contacts

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/contact-role	
Requirement	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/contact-

Req 55	role
Test Method	Validate within the XML Instance that either *xlink:arcrole* or *xlink:role* attribute is present in the *member* element of *ContactList*.
Test Type	Capability

B.1.12 The *typeOf* property shall provide the uniqueID and resolvable location of the description on the referenced object

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/type-of-reference	
Requirement Req 56	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/type-of-reference
Test Method	Validate within the XML Instance that an *xlink:title* and *xlink:href* are present in any *typeOf* element of an XML Instance.
Test Type	Capability

B.1.13 The feature of interest property shall specify a role, and if available, the uniqueID of the feature

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/foi-arcrole-and-title	
Requirement Req 57	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/foi-arcrole-and-title
Test Method	Validate within the XML Instance that the *member* property of a *FeatureList* has a *xlink:arcrole* attribute present.
Test Type	Capability

B.1.14 The *definition* attribute required for *ObservableProperty*

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/observable-definition	
Requirement	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/observable-

Req 58	definition
Test Method	Validate within the XML Instance, the presence and resolvability of the *definition* attribute for any *ObservableProperty* element.
Test Type	Capability

B.1.15 Use aggregate data for related data elements

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/data-record	
Requirement Req 59	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/data-record
Test Method	Validate by inspection that dependent data components and those representing the state at a given time are encapsulted within an appropriate aggregate data component.
Test Type	Capability

B.1.16 Use Vector for *inputs*, *outputs*, and *parameters* that specify position

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/vector-use	
Requirement Req 60	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/vector-use
Test Method	Validate that positions within *inputs*, *outputs*, and *parameters* utilize the *swe:Vector* element.
Test Type	Capability

B.1.17 Use of resolvable URL to reference data streams

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/data-stream-url	
Requirement Req 61	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/data-stream-url

Test Method	Validate the presence of an *xlink:href* atribute with resolvable URL for *DataStream values* provided by reference.
Test Type	Capability

B.1.18 Use *DataChoice* in multiplexed data streams

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process/multiplexed-data-stream	
Requirement Req 62	http://www.opengis.net/spec/sensorml/2.0/req/xml/core-process/multiplexed-data-stream
Test Method	Validate that a *DataStream* consisting of disparate packages utilizes *DataChoice* to encapsulate those packages and that each package is defined as an *item*.
Test Type	Capability

B.2 Conformance Test Class: Simple Process Schema

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/simple-process	
Target Type	XML Instance
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/simple-process**
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process**

All tests in this conformance test class and in the following shall be used to check conformance of XML instances created according to the schemas defined in this standard. They shall also be used to check conformance of software implementations that output XML instances.

B.2.1 Compliance with simple_process XML schemas and Schematron patterns

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/simple-process/schema-valid	
Requirement Req 63	http://www.opengis.net/spec/sensorml/2.0/req/xml/simple-process/schema-valid
Test Method	Validate the XML instance containing core process with the "simple_process.xsd" XML schema file and the Schematron patterns in "simple_process.sch".
Test Type	Capability

B.3 Conformance Test Class: Aggregate Process Schema

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process	
Target Type	XML Instance
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/aggregate-process**
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/xml/simple-process**

All tests in this conformance test class and in the following shall be used to check conformance of XML instances created according to the schemas defined in this standard. They shall also be used to check conformance of software implementations that output XML instances.

B.3.1 Compliance with simple_process XML schemas and Schematron patterns

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/schema-valid	
Requirement Req 64	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/schema-valid
Test Method	Validate the XML instance containing core process with the "aggregate_process.xsd" XML schema file and the Schematron patterns in "aggregate_process.sch".
Test Type	Capability

B.3.2 Title and resolvable URL required for components provided by Reference

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/component-reference	
Requirement Req 65	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/component-reference
Test Method	Validate the presence of *xlink:title* and *xlink:href* for the *component* property when its value is provided by reference.
Test Type	Capability

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/input-connection-restrictions	
Requirement Req 66	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/input-connection-restrictions
Test Method	Validate that there are no input-to-input connections, other than from the aggregate process to one or more of its components.
Test Type	Capability

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/output-connection-restrictions	
Requirement Req 67	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/output-connection-restrictions
Test Method	Validate that there are no output-to-output connections, other than from a component process to the output of the aggregate process to.
Test Type	Capability

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/multiple-connections	
Requirement Req 68	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/multiple-connections
Test Method	Validate that there are no multiple connection within any given input port.
Test Type	Capability

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/parameter-connection-restrictions	
Requirement Req 69	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/parameter-connection-restrictions

Test Method	Validate that there are no parameters listed as a source in any path.
Test Type	Capability

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/property-connection-restrictions	
Requirement Req 70	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/property-connection-restrictions
Test Method	Validate that no properties other than inputs, outputs, and parameters are included as a destination.
Test Type	Capability

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process/input-connection-restrictions	
Requirement Req 71	http://www.opengis.net/spec/sensorml/2.0/req/xml/aggregate-process/input-connection-restrictions
Test Method	Validate that all paths lead to valid ports.
Test Type	Capability

B.4 Conformance Test Class: Physical Component Schema

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component	
Target Type	XML Instance
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-component**
Dependency	**Http://www.opengis.net/spec/sensorml/2.0/conf/xml/simple-process**

All tests in this conformance test class and in the following shall be used to check conformance of XML instances created according to the schemas defined in this standard. They shall also be used to check conformance of software implementations that output XML instances.

B.4.1 Compliance with physical_component XML schemas and Schematron patterns

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-system/schema-valid	
Requirement Req 72	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-system/schema-valid
Test Method	Validate the XML instance containing core process with the "physical_component.xsd" XML schema file and the Schematron patterns in "physical_component.sch".
Test Type	Capability

B.4.2 A physical process can only attach to a physical process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component/attached-to-target	
Requirement Req 73	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/attached-to-target
Test Method	Validate that the *xlink:href* attribute for the *attachedTo* property resolves to a *PhysicalComponent* or *PhysicalSystem*.
Test Type	Capability

B.4.3 The *attachedTo* element shall have *xlink:title* and *xlink:href*

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component/attached-to-reference	
Requirement Req 74	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/attached-to-reference
Test Method	Validate that the *attachedTo* property has values for the *xlink:href* and *xlink:title* attributes.
Test Type	Capability

B.4.4 Position requires a DataRecord with two Vectors

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component/position-by-position	
Requirement Req 75	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/position-by-position
Test Method	When the *position* element takes a *swe:DataRecord* as its value, validate that the *DataRecord* contains two *swe:Vector* elements as its fields.
Test Type	Capability

B.4.5 Dynamic state requires a Data Array or Process

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component/dynamic-state	
Requirement Req 76	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/dynamic-state
Test Method	Validate through inspection that time-tagged locations, positions, or state values are provided by either *swe:DataArray* or a class derived from *sml:AbstractProcess*.
Test Type	Capability

B.4.6 Trajectory requires a DataArray with a time field and one or more Vectors

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component/position-by-trajectory	
Requirement Req 77	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/position-by-trajectory
Test Method	When the *position* element takes a *swe:DataArray* as its value, validate that the *DataArray* contains a time field and one or more *swe:Vector* elements as its fields.
Test Type	Capability

B.4.7 Process required for positions or state provided on-demand

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component/position-by-process	
Requirement Req 78	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-component/position-by-process
Test Method	When the *position* element takes an instance of *sml:AbstractProcess* as its value, validate that the output of the process contains a *swe:DataArray* with time-tagged trajectory data, or a *swe:DataRecord* with time-tagged position or state data.
Test Type	Capability

B.5 Conformance Test Class: Physical System Schema

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-system	
Target Type	XML Instance
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/physical-system**
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/xml/aggregate-process**
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-component**

All tests in this conformance test class and in the following shall be used to check conformance of XML instances created according to the schemas defined in this standard. They shall also be used to check conformance of software implementations that output XML instances.

B.5.1 Compliance with physical_system XML schemas and Schematron patterns

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/physical-system/schema-valid	
Requirement Req 79	http://www.opengis.net/spec/sensorml/2.0/req/xml/physical-system/schema-valid
Test Method	Validate the XML instance containing core process with the "physical_system.xsd" XML schema file and the Schematron patterns in "physical_system.sch".
Test Type	Capability

B.6 Conformance Test Class: Configurable Process Schema

Conformance Test Class	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/configuration	
Target Type	XML Instance
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/model/configuration**
Dependency	**http://www.opengis.net/spec/sensorml/2.0/conf/xml/core-process**

All tests in this conformance test class and in the following shall be used to check conformance of XML instances created according to the schemas defined in this standard. They shall also be used to check conformance of software implementations that output XML instances.

B.6.1 Compliance with configuration XML schemas and Schematron patterns

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/configuration/schema-valid	
Requirement Req 80	http://www.opengis.net/spec/sensorml/2.0/req/xml/configuration/schema-valid
Test Method	Validate the XML instance containing core process with the "configuration.xsd" XML schema file and the Schematron patterns in "configuration.sch".
Test Type	Capability

B.6.2 Modes can change values of parameters

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/configuration/mode-restriction	
Requirement Req 81	http://www.opengis.net/spec/sensorml/2.0/req/xml/configuration/mode-restriction
Test Method	Validate that *setValue* references only *parameter* properties and only *parameter* properties within the current process or parent process.
Test Type	Capability

B.6.3 Modes can only set values of parameters to those allowed by its constraints

Conformance Test	
http://www.opengis.net/spec/sensorml/2.0/conf/xml/configuration/parameter-values	
Requirement Req 82	http://www.opengis.net/spec/sensorml/2.0/req/xml/configuration/parameter-values
Test Method	Validate that the *parameter* values set within *Mode* are within the appropriate range as defined in the *swe:AllowedValues* property of that *parameter*.
Test Type	Capability

Annex C
(normative)

UML to XML Schema Encoding Rules

This standard follows a model-driven approach to automatically generate the XML Schema detailed in Section 8 from the UML models introduced in Section 7. The encoding rules used by this standard to generate XML schema are derived from GML encoding rules defined in ISO 19136.

A few changes have been introduced to GML encoding rules in order to accommodate for Sensor Web Enablement specific needs. These changes are listed and explained below:

- Relaxed rule on the mandatory gml:id attribute. gml:id is thus optional in SWE schemas.
- Introduced new stereotype for soft-typed-properties.
- Added support for encoding simple-type properties as XML attributes.
- Use different base type for <<Type>> stereotype (Elements are derived from anyType and made substitutable for gml:AbstractValue instead of gml:AbstractGML).

Annex D: Revision History

Date	Release	Author	Paragraph modified	Description
2012-03-16	2.0 draft	Mike Botts	All	Initial reviewed version
2012-07-27	2.0	Mike Botts	All	Completed specification
2012-09-12	2.0	John Greybeal	All	Edits and corrections throughout
2012-09-12	2.0	Alex Robin	All	Edits and corrections throughout
2013-07-04	2.0	Mike Botts	All	Final draft version
2013-10-29	2.0	Mike Botts	All	Editorial and requirements
2013-12-03	2.0	Mike Botts/Simon Cox	All	Final updates to model images and requirements references