Open Geospatial Consortium

Submission Date: 2017-08-31

Approval Date: 2017-09-15

Publication Date: 2018-01-18

External identifier of this OGC® document: http://www.opengis.net/doc/BP/SWE-JSON/1.0

Internal reference number of this OGC® document: 17-011r2

Version: 1.0

Category: OGC® Best Practice

Editor: Alex Robin

JSON Encoding Rules

SWE Common / SensorML

Warning

This document defines an OGC Best Practice on a particular technology or approach related to an OGC standard. This document is **not** an OGC Standard and may not be referred to as an OGC Standard. This document is subject to change without notice. However, this document is an **official** position of the OGC membership on this particular technology topic.

Document type: OGC® Best Practice
Document subtype:
Document stage: Approved for public release
Document language: English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Contents

i. Abstract

This document describes new JavaScript Object Notation (JSON) encodings for the Sensor Web Enablement (SWE) Common Data Model and the Sensor Model Language (SensorML). Rather than creating new JSON schemas, this document defines encoding rules that allow auto-generation of JSON instances that conform to the Unified Modeling Language (UML) models. Alternatively, the mappings given in the second part of the document can be used to convert bi-directionally between XML and JSON representations.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, ogc document, sensor, actuator, swe, json, extension, encoding, encoding rules, data model, sensor model, sensorml, swe common

iii. Preface

The SensorML standard [OGC 12-000] defines conceptual models and a corresponding XML encoding (specified by an XML schema) for describing sensors, actuators and associated processing components. Likewise, the SWE Common Data Model standard [OGC 08-194-r1], which is also a dependency of the SensorML standard, defines conceptual models and XML encodings for describing the structure and efficiently encoding values of data streams.

This document describes new JSON encoding rules that are applicable to both standards, and enable bi-directional conversion between XML and JSON data representations of the conceptual models. The rules are designed to be easy to implement so that software that already supports the XML encoding can be easily updated to support the new JSON representation.

Suggested additions, changes, and comments on this document are welcome and encouraged. Such suggestions may be submitted by email message, or by making suggested changes in an edited copy of this document.

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

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

iv. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Botts Innovative Research, Inc.
- Sensia Software LLC

v. Submitters

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

Name	Affiliation
Alexandre Robin	Sensia Software LLC
Michael E. Botts	Botts Innovative Research, Inc.

vi. Future work

This document only defines JSON encoding rules for the SensorML and SWE Common Data Model standards, but the same rules could potentially be applied to other OGC standards. In particular, the following standards of the Sensor Web Enablement initiative could be aligned to use the same rules:

- Observations and Measurements (O&M) JSON: these recently proposed encodings could potentially be revisited so that the exact same rules are used to derive the JSON representation.
- SensorThings API: this RESTful interface standard could make use of the SensorML JSON encoding proposed herein. A future v2.0 could also be aligned to use the proposed SWE Common JSON representation rather than the slightly different models/JSON structure that was created for v1.0.
- In general, there is a need for a set of rules for transforming UML into JSON beyond SWE. These rules are needed for most of the OGC standards including OWS Common.

There are also some aspects of the encoding rules that should probably be debated:

- Should GeoJSON be used whenever a GML geometry is used?
- Should plural words be used for properties with multiplicity greater than one encoded as arrays?

The authors have endeavored to ensure that the encoding rules are compatible with those identified by Testbed-12 [OGC 16-051]. However, aspects such as namespaces are not supported in the current version of the encoding rules. It is envisioned that future versions of the encoding rules will apply lessons from Testbed-12 regarding use of JSON Schema or JSON-LD for supporting namespaces.

1. Scope

This document describes encoding rules that can be used to derive a JSON structure (i.e. JSON schema) from the UML conceptual models of SWE Common and SensorML. It also describes direct mappings between XML and JSON instances to ease implementation of software that does conversion between XML and JSON representations.

2. References

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

OGC 12-000: OGC® SensorML: Model and XML Encoding Standard, version 2.0, February 2014

OGC 08-094r1: OGC® SWE Common Data Model Encoding Standard, version 2.0, January 2011

OGC 07-036: Geographic Markup Language, version 3.2.1, August 2007, Annex E: UML-to-GML application schema encoding rules

IETF RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format, March 2014

OGC 16-051: Testbed-12: Javascript-JSON-JSON-LD Engineering Report, May 2017

3. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

For the purposes of this document, no other terms and definitions apply.

4. Conventions

4.1 Abbreviated Terms

JSON	JavaScript Object Notation
O&M	Observations and Measurements
SensorML	Sensor Model Language
SWE Common	SWE Common Data Model
XML	eXtensible Markup Language

4.2 Identifiers

The normative provisions in this specification are denoted by the URI:

http://www.opengis.net/spec/SWE-JSON/1.0

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

5. JSON Encoding Rules

5.1 JSON Information Model

In this document, the choice was made to maintain the complete information model in the JSON representation, and to not apply any simplification. With the exception of namespaces, all information included in the UML conceptual models is thus present in the JSON representation (similarly to the XML encoding). This allows for bi-directional conversion between XML and JSON without any loss of information.

The encoding rules defined in this document refer to JSON data types whose definitions are recalled below:

Objects: An object structure is represented as a pair of curly brackets surrounding zero or more name/value pairs (or members). Members are separated by commas. Each member must have a distinct name (i.e. a JSON object is essentially a map).

Arrays: An array structure is represented as square brackets surrounding zero or more values (or elements). Elements are separated by commas.

Numbers: A decimal or integer number represented in base 10, with a sign and optional exponent.

Strings: A string of Unicode characters that begins and ends with quotation marks.

5.2 Requirements Class: UML - JSON Mapping

Requirements Class	
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-json	
Target Type	JSON Instance

This requirements class defines bi-directional mappings between the UML conceptual models for SWE Common/SensorML and the proposed JSON encoding.

5.2.1 UML Stereotypes

SWE Common and SensorML conceptual models use many of the UML conventions defined in annex E of [OGC 07-036]. In particular, the following stereotypes are used: **<<Type>>, <<DataType>>** and **<<property>>**.

In addition, some UML tags are used to customize encoding of certain properties:

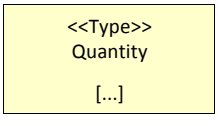
- **by-reference**: indicates that the property value can be given by reference (with the XML encoding, this translates into xlink attributes on the property)
- **soft-typed**: indicates that the property is soft-typed and must carry a "name" attribute to be fully qualified.

5.2.2 Rules for UML Classes

Requirement	
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-class-type	
Req 1	An instance of a UML class with stereotype <<Type>> or <<DataType>> or no stereotype at all shall be encoded as a JSON object containing a "type" member whose value is the name of the UML class.

Example 1: Object type

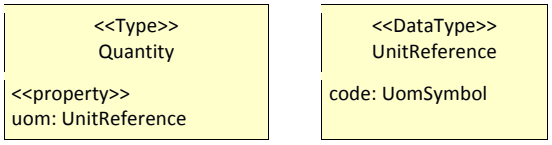
UML	**JSON**

	{
	"type": "Quantity",
	[...]
	}

Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-class-ref
Req 2 An instance of a UML class derived from the *Reference* class shall be encoded as a JSON object without a “type” member.

Example 2: Reference object

UML	JSON

	{
	“type”: “Quantity”,
	“uom”: { “code”: “hPa” }
	}

5.2.3 Rules for UML Properties

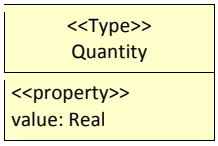
Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property
Req 3 A UML attribute or association with stereotype <<property>> shall be encoded as a JSON object member. The member’s name shall be the same as the UML attribute or association’s name.

Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property-single
Req 4 If the UML property has multiplicity one, the value of the JSON member shall be the property instance value.

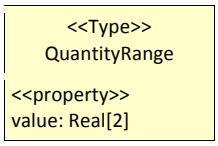
Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property-multi
Req 5 If the UML property has multiplicity greater than one, the value of the JSON member shall be a JSON array containing a list of property instance values.

Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property-numeric
Req 6 If the UML property is of type “Real” or “Integer”, its instance value shall be a JSON number.

Example 3: Property with numeric type

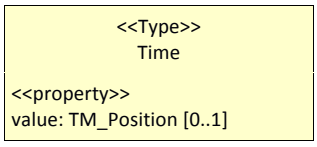
UML	JSON
	{ "type": "Quantity", "value": 3.2 }

Example 4: Property with numeric type and multiplicity > 1

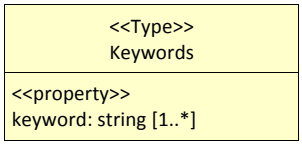
UML	JSON
	{ "type": "Quantity", "value": [3.2, 10.4] }

Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property-string
Req 7 If the UML property is **not** of type "Real" or "Integer", its instance value shall be a JSON string.

Example 5: Property with non-numeric type

UML	JSON
	{ "type": "Time", "value": "2014-05-02T23:45:32Z" }

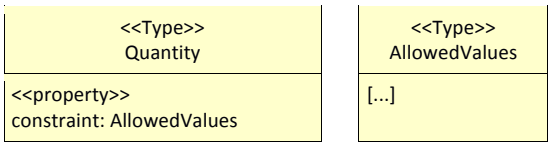
Example 6: Property with non-numeric type and multiplicity > 1

UML	JSON
	{ "type": "Keywords", "keyword": ["word1", "word2"] }

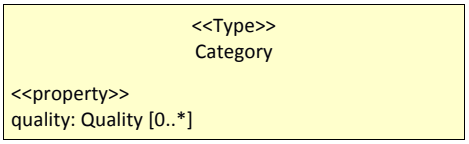
Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property-complex

Req 8 If the UML property is of object type, its instance value shall be the JSON object corresponding to the UML class (see Req 1).

Example 7: Property with object type

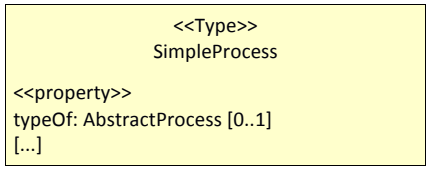
UML	JSON
	{ "type": "Quantity", "constraint": { "type": "AllowedValues", ... } }

Example 8: Property with object type and multiplicity > 1

UML	JSON
	{ "type": "Category", "quality": [{ ... }, { ... }] }

Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property-ref
Req 9 If a UML property allows the "by-reference" pattern as defined in [ISO 19136] and a reference is used, its instance value shall be a JSON object with a "href" member and optional "role" and "arcrole"members.

Example 9: By-reference property

UML	JSON
	{ "type": "SimpleProcess", "typeOf": { "href": "http://myuri" }, ... }

Example 10: By-reference property with multiplicity > 1

UML	JSON

<<Type>>
Quantity
 	
<<property>>
field: AbstractDataComponent [1..*]
[...]
	{
	"type": "DataRecord",
	"field": [
	{ "href": "http://myuri" },
	{ ... }
	]
	}

Requirement
http://www.opengis.net/spec/SWE-JSON/1.0/req/uml-property-softtyped
Req 10 If the UML property is a "soft-typed" property as defined in [OGC 08-094r1], a "name" member is added to the property instance value object.

Example 11: Soft-typed property

UML	**JSON**

<<Type>>
DataRecord
 	
<<property>>
field: AbstractDataComponent [1..*]
[...]
	{
	"type": "DataRecord",
	"field": [
	{ "name": "temp", ... },
	{ "name": "press", ... }
	]
	}

5.3 XML - JSON Mapping (informative)

This section describes bi-directional mappings between the SWE Common/SensorML XML representation and the proposed JSON encoding. This is not normative but can help updating software that already support the XML representation.

Although these mappings are expressed in terms of XML and JSON structure, it is important to understand that an implementation of these mappings does NOT imply generating an actual XML document before it can be converted to JSON and vice-versa. Implementation can instead apply the mapping "on-the-fly" to whatever representation (e.g. programming language structures such as objects) they desire.

5.3.1 XML Elements Types

XML schemas for SWE Common and SensorML were auto-generated from UML models using well defined encoding rules; thus they are consistent in the way they alternate two types of XML elements:

- Object elements start with an upper-case letter and correspond to class names in the conceptual models. Object elements can contain several property elements.

- Property elements start with a lower-case letter and correspond to a class attributes names in the conceptual models. A property element can contain either an inline value or a single nested object element.

These two types of XML elements are encoded differently in JSON.

5.3.2 Namespaces

JSON does not have the concept of namespaces natively, thus none are used when representing the SWE Common and SensorML conceptual models as JSON in this current version of the encoding rules.

5.3.3 XML Object Elements Mappings

An XML object element that is the root of the document is encoded as a JSON object.

An XML object element that is not the root of the document is necessarily nested within a parent XML property element. Since XML complex property elements in SWE schemas always take a single child element, we collapse both the property and its complex value in a single JSON object. In this case, only its local name is retained and set as value of the "type" member of the enclosing JSON object.

Example 12: Object element as root

XML	JSON

<Quantity>
[...]
</Quantity>
	{
	"type": "Quantity",
	...
	}

Example 13: Object element as child

XML	JSON

<prop>
<Quantity>
[...]
</Quantity>
</prop>
	"prop": {
	"type": "Quantity",
	...
	}

5.3.4 XML Property Elements Mappings

An XML property element maps to a JSON object's member. The member's name corresponds to the local part (i.e. without the namespace prefix) of the XML element's name.

5.3.4.1 Property with simple type (multiplicity = 1)

If the XML property has multiplicity one and is of simple type (i.e. the property contains an inline value), the value of the JSON member is a JSON number or string.

Example 14: Simple property value

XML	JSON

<swe:Quantity>
<swe:value>3.2</swe:value>
</swe:Quantity>
	{
	"type": "Quantity",
	"value": 3.2
	}

5.3.4.2 Property with complex type (multiplicity = 1)

If the XML property has multiplicity one and is of complex type (i.e. the property contains a nested XML element or attribute), the value of the JSON member is a JSON object.

Example 15: Property with nested attribute

XML	JSON

<swe:Quantity>
<swe:uom code="hPa"/>
</swe:Quantity>
	{
	"type": "Quantity",
	"uom": { "code": "hPa" }
	}

Example 16: Property with nested element

XML	JSON

<swe:Quantity>
<swe:constraint>
<swe:AllowedValues>
[...]
</swe:AllowedValues>
</swe:constraint>
</swe:Quantity>
	{
	"type": "Quantity",
	"constraint": {
	"type": "AllowedValues",
	...
	}
	}

5.3.4.3 Property with simple type (multiplicity > 1)

If the XML property has multiplicity greater than one and is of simple type, the value of the JSON member is a JSON array that collects all values from the different occurrences of this property.

Example 17: Simple property with multiplicity > 1

XML	JSON

<sml:Keywords>
<sml:keyword>word1</sml:keyword>
<sml:keyword>word2</sml:keyword>
</sml:Keywords>
	{
	“type”: “Keywords”,
	“keyword”: [“word1”, “word2”]
	}

5.3.4.4 Property with complex type (multiplicity > 1)

If the XML property has multiplicity greater than one and is of complex type, the value of the JSON member is a JSON array that collects all values from the different occurrences of this property as JSON objects.

Example 18: Complex property with multiplicity > 1

XML	JSON

<swe:DataRecord>
<swe:field name=”temp”>[...]</swe:field>
<swe:field name=”press”>[...]</swe:field>
</swe:DataRecord>
	{
	“type”: “DataRecord”,
	“field”: [
	{ “name”: “temp” },
	{ “name”: “press” }
	]
	}

5.3.5 XML Attributes Mappings

An XML attribute maps to a JSON object’s member whose value is a JSON number or string. The member’s name corresponds to the local part (i.e. without the namespace prefix) of the XML attribute’s name.

Example 19: Attribute on property element

XML	JSON

<swe:field name=”temp”>
[...]
</swe:field>
	“field”: {
	“name”: “temp”,
	...
	}

An XML attribute that is a child of an XML object element is appended directly to the enclosing JSON object after the “type” object member.

Example 20: Attribute on object element

XML	JSON

<swe:Quantity definition=”uri”/>	{
	“type”: “Quantity”,
	“definition”: { “code”: “hPa” }
	}

5.3.6 XML Data Types Mappings

An element or attribute value of type “decimal”, “float” or “double” or any type derived from these is encoded as a JSON number.

An element or attribute value of any other type is encoded as a JSON string.

XML values that are fixed in the schema can be omitted in JSON.

5.4 Comparative XML and JSON Examples

The following snippets show an XML instance and the corresponding JSON obtained by applying the mappings defined in this document.

5.4.1 SWE Common Example

XML

```
<DataRecord>
  <label>Weather Data Record</label>
  <description>Record of synchronous weather measurements</description>
  <field name="ts">
    <Time definition="http://www.opengis.net/def/property/OGC/0/SamplingTime"
        referenceFrame="http://www.opengis.net/def/trs/OGC/0/GPS">
      <label>Sampling Time</label>
      <uom href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
    </Time>
  </field>
  <field name="temp">
    <Quantity definition="http://mmisw.org/ont/cf/parameter/air_temperature">
      <label>Air Temperature</label>
      <uom code="Cel"/>
    </Quantity>
  </field>
  <field name="press">
    <Quantity definition="http://mmisw.org/ont/cf/parameter/air_pressure">
      <label>Air Pressure</label>
      <uom code="mbar"/>
    </Quantity>
  </field>
</DataRecord>
```

JSON

```
{
  "type": "DataRecord",
  "label": "Weather Data Record",
  "description": "Record of synchronous weather measurements",
  "field": [
    {
      "name": "ts",
      "type": "Time",
      "definition": "http://www.opengis.net/def/property/OGC/0/SamplingTime",
      "referenceFrame": "http://www.opengis.net/def/trs/OGC/0/GPS",
      "label": "Sampling Time",
      "uom": { "href": "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian" }
```

```
  },
  {
    "name": "temp",
    "type": "Quantity",
    "definition": "http://mmisw.org/ont/cf/parameter/air_temperature",
    "label": "Air Temperature",
    "uom": { "code": "Cel" }
  },
  {
    "name": "press",
    "type": "Quantity",
    "definition": "http://mmisw.org/ont/cf/parameter/air_pressure",
    "label": "Air Pressure",
    "uom": { "code": "mbar" }
  }
 ]
}
```

5.4.2 SensorML Example

XML

```
<PhysicalComponent id="MY_SENSOR">
 <description>Thermometer on the window of the Cass Building, Room 315</description>
 <identifier>urn:icd:stations:FR8766</identifier>
 <identification>
  <IdentifierList>
   <identifier>
    <Term definition="http://sensorml.com/ont/swe/property/ShortName">
     <label>Short Name</label>
     <value>Thermometer FR8766</value>
    </Term>
   </identifier>
   <identifier>
    <Term definition="http://sensorml.com/ont/swe/property/Manufacturer">
     <label>Manufacturer Name</label>
     <value>ACME Inc</value>
    </Term>
   </identifier>
   <identifier>
    <Term definition="http://sensorml.com/ont/swe/property/SerialNumber">
     <label>Serial Number</label>
     <value>FT5743456566-997</value>
    </Term>
   </identifier>
  </IdentifierList>
 </identification>
 <outputs>
  <OutputList>
   <output name="temp">
    <Quantity definition="http://sweet.jpl.nasa.gov/2.2/quanTemperature.owl#Temperature">
     <uom code="Cel"/>
    </Quantity>
   </output>
  </OutputList>
 </outputs>
 <position>
  <Point id="stationLocation">
   <srsName>http://www.opengis.net/def/crs/EPSG/0/4326</srsName>
   <srsDimension>2</srsDimension>
   <pos>47.8 88.56</pos>
```

```
   </Point>
  </position>
</PhysicalComponent>
```

JSON


```
{
 "type": "PhysicalComponent",
 "id": "MY_SENSOR",
 "description": "Thermometer on the window of the Cass Building, Room 315",
 "identifier": "urn:icd:stations:FR8766",
 "identification": [
  {
   "type": "IdentifierList",
   "identifier": [
    {
     "type": "Term",
     "definition": "http://sensorml.com/ont/swe/property/ShortName",
     "label": "Short Name",
     "value": "Thermometer FR8766"
    },
    {
     "type": "Term",
     "definition": "http://sensorml.com/ont/swe/property/Manufacturer",
     "label": "Manufacturer Name",
     "value": "ACME Inc"
    },
    {
     "type": "Term",
     "definition": "http://sensorml.com/ont/swe/property/ModelNumber",
     "label": "Manufacturer Model",
     "value": "T911"
    },
    {
     "type": "Term",
     "definition": "http://sensorml.com/ont/swe/property/SerialNumber",
     "label": "Serial Number",
     "value": "FT5743456566-997"
    }
   ]
  }
 ],
 "classification": [
  {
   "type": "ClassifierList",
   "classifier": [
    {
     "type": "Term",
     "definition": "http://sensorml.com/ont/swe/property/IntendedApplication",
     "label": "Intended Application",
     "value": "Atmospheric Temperature"
    }
   ]
  }
 ],
 "outputs": {
  "type": "OutputList",
  "output": [
   {
    "name": "temp",
    "type": "Quantity",
    "definition": "http://sweet.jpl.nasa.gov/2.2/quanTemperature.owl#Temperature",
    "uom": { "code": "Cel" }
   }
  ]
 },
 "position": [
  {
   "type": "Point",
   "id": "stationLocation",
```

```
    "srsName": "http://www.opengis.net/def/crs/EPSG/0/4326",
    "srsDimension": "2",
    "pos": "47.8 88.56"
   }
  ]
}
```

6. Existing Implementation

An implementation of the proposed JSON encodings is available as part of the OpenSensorHub project: http://www.opensensorhub.org.

In order to reuse all code from the existing implementation of SWE Common and SensorML, the XML to/from JSON encoding mappings were implemented directly against the Streaming API for XML (StAX) which is available in Java. StAX is a very efficient streaming API for XML available in Java and OSH uses it to serialize/deserialize Java objects to/from OGC XML representations at high speed and with low memory footprint (i.e. the use of StAX means there is no need to create a Document Object Model (DOM) representation of the XML document in memory).

OSH adds implementations of StAX interfaces (namely *XMLStreamReader* and *XMLStreamWriter)* called *JsonStreamReader* and *JsonStreamWriter* that can read/write JSON directly and send/consume data directly using the StAX pipeline. This is a very efficient approach because no conversion between JSON and XML actually occurs, rather the data encoded in either format is consumed directly into the application through the same StAX pipeline and, consequently, JSON encoders/decoders can be used interchangeably with their XML counter parts whenever needed.

A demonstration SOS server supporting the JSON encoding for various datasets has been setup and example requests are available at the following location:

http://sensiasoft.net:8181/demo.html