Open Geospatial Consortium

Publication Date: 2015-08-19

Approval Date: 2015-06-04

Posted Date: 2015-05-31

Reference number of this document: OGC 15-053r1

Reference URL for this document: http://www.opengis.net/doc/PER/tb-11-geojson-in-ogc

Category: Public Engineering Report

Editor: Joan Masó

Testbed 11 Implementing JSON/GeoJSON in an OGC Standard Engineering Report

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

Document type: OGC® Engineering Report
Document subtype: NA
Document stage: Approved for public release
Document language: English

License Agreement

Contents

Page

Figures

Page

Tables

Page

Abstract

In the OGC Testbed 11, the Cross-Community Interoperability (CCI) thread had a key objective of building on the work accomplished in the OGC 8, 9 and 10 Testbeds. The goal of the CCI threads is to increase interoperability between communities sharing geospatial data. This thread made advances in semantic mediation approaches for data discovery, access and use of heterogeneous data models and heterogeneous metadata models. This particular Engineering Report (ER) is part of the OGC efforts to advance the OGC Architecture with the adoption of REST interfaces and more encodings such as JSON.

This document is a response to a recommendation expressed in OGC 14-113 OGC JSON position statement of including JSON/GeoJSON research as a component in the OGC Testbed 11 activity with the goals:

- Develop a consistent approach across the OGC suite of service standards for using JSON and GeoJSON.

- Define and document rules for JSON and GeoJSON extensions to OGC Web Service encodings, which was started in Testbed 10 and reflected in the draft OWS Context GeoJSON Encoding.

The document covers several aspects of JSON and its relation with OGC standards:

- Clause 5 gives a general introduction to JSON concepts that will be the bases for the next clauses (e.g.: JSON schema, JSON-LD, GeoJSON, TopoJSON, etc).

- Clause 6 revisits the rules provided in OGC 14-009r2 OGC Testbed-10 Rules for JSON and GeoJSON Adoption: Focus on OWS-Context and complementing them with missing aspects. In particular, Subclause 6.1.2.2 is proposing strategies for implementing links in JSON.

- Clause 7 reviews GeoJSON. Limitations of GeoJSON are enumerated and simple solutions are suggested in subclause 5.5.1 but not discussed in depth. This ER considers that such issues need to be resolved by the community. For that reason this ER accepts GeoJSON as is: an encoding for simple features. It concentrates on making GeoJSON more solid by associating it with to a JSON schema. In subclause 7.3 and 7.4 a major issue with the coordinates that prevents an easy adoption of JSON-LD in GeoJSON is identified and a solution is proposed based on Well Known Text. The section proposes a way to connect the simple features and WFS to RDF and linked data by introducing geospatial rules in common JSON-LD parsers.

- Clause 8 is a contribution to the current work in the OWS Context Standards Working Group (SWG). A JSON schema for OWS Context is proposed and an analyses on how JSON-LD can be applied to OWS Context JSON is provided. This clause also proposes another encoding for OWS Context that is based on Microdata and HTML5 that can also be expressed in JSON-LD using schema.org approach.

- Clause 9 proposes a JSON encoding for coverages based on recoding GMLCov in JSON. In this part a small demo has also elaborated (http://www.creaf.uab.cat/joanma/coveragejson/) to demonstrate the feasibility of a map browser in HTML5 based on WCS instead of WMS.

- Clause 10 makes proposals on how JSON can be used in OWS services. The section starts by making recommendations on how to encode GetCapabilities in JSON and then reviews some OWS standards such as WMS, WMTS and WFS. Some final recommendation for Geospatial User Feedback are also provided.

- Clause 11 proposes a set of rules to translate UML into JSON-LD. The rules are compendium of best practices that are illustrated throughout this document and are presented in a single place for convenience.

Business Value

The incorporation of JSON in OGC standards has been demanded requirement from the geospatial developer community for some time. They see an opportunity to increment productivity in the web browser based applications and in the mobile phone apps sector. Even if, in theory, users should not impacted by a migration from XML to JSON, in practice the simplification in the application developments will give more “free time” to developers that can spend in providing better solutions to users (such as more interactivity or more functionality). Developers will also be able to create code that is easier to maintain and as a consequence be more error free. The join adoption of REST + JSON will increase interoperability with the non geospatial world and re-stimulate the creation of new much-ups (a concept that was in fact introduced in the XML+AJAX years) and increase the use of geospatial technology. The fact that new emerging open source map browsers such as Leaflet (and MapBox, and CartoBD) have adopted GeoJSON as their main encoding and Esri has their RESTful API are signs that there is a business model that we in the OGC cannot ignore

The document includes some concrete proposals that demonstrate how OGC standards can be combined with adopted mass market standards In particular both OWS Context in Microdata (complemented with the JSON-LD encoding for schema.org) and Geospatial

User Feedback standards could increasing interoperability by making the geospatial information equivalent to any other product that is exposed an documented in the web using structured content strategies recommended in schema.org (e.g. Movies, Recipes, Products, Reviews). We recommend to work with the main search engine actors to increase discoverability of geospatial information; a recurrently mentioned problem by the geospatial community.

This document also proposes a way to use a JSON version of GMLCov that could be used in combination with WCS to present data directly in web browsers (instead of presenting pictorial representations). This should increase usability and productivity of rectified grid coverage data.

Keywords

ogcdocs, ogc documents, testbed-11, encoding, JSON, GeoJSON, TopoJSON, JSON-LD, RDF, HTML5, Well-Known-Text, WKT, GMLCov, WFS-JSON, WMS, WMTS, GUF, Microdata, OWS Context.

Implementing JSON/GeoJSON in an OGC Standard: Testbed 11 Engineering Report

1 Introduction

1.1 Scope

This OGC® Engineering Report (ER) provides guidelines for the use of JSON in OGC standards for encoding requests and responses of services. This ER also provides guidance on how to use GeoJSON in the OGC context. In addition guidelines for the use of JSON-LD and JSON schema in several OGC standards are provided.

This ER is applicable to OGC service standards in general and in particular to the ones that deal with encoding features such as WFS but also proposes additions to WCS, WMS and WMTS.

1.2 Document contributor contact points

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

Name	Organization
Joan Masó	UAB-CREAF

Special thanks to Jon Harry and Peter Vretanos for long email discussions and to Josh Lieberman for its leadership in the Testbed-11 CCI group and contributions.

1.3 Future work

Improvements in this document are desirable to with the help of other Testbed 11 participants and the rest of the TC. The document also makes some recommendations on possible additions to current standards. Experimenting with these additions will complement this work.

This document has not elaborated on the security issues. 14-113 OGC JSON Position Statement document mentions some security issues related with the JavaScript interpreter

to execute JSON text dynamically as embedded JavaScript and the possibility of inserting malicious code that need further consideration.

The document partially addresses the lack of an agreed method to validate semantically in JSON and possible alternatives for schema documents. The authors are particularly sensible to semantic and syntactic validation and most of the code shown in this document has been validated as much as current technologies allow it. This document proposes some alternatives and recommendations but a formal and clear solution is still needed.

1.4 Forward

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.

2 References

The following documents are referenced in 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 06-121r9, *OGC® Web Services Common Standard, Version 2.0*

NOTE This OWS Common Standard contains a list of normative references that are also applicable to this Implementation Standard.

IETF RFC 4627, The application/json Media Type for JavaScript Object Notation (JSON), D. Crockford, http://www.ietf.org/rfc/rfc4627.txt

OGC 06-103r4, OGC Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture v.1.2.1

OGC 14-009r1, OGC Testbed-10 Rules for JSON and GeoJSON Adoption: Focus on OWS-Context, Pedro Gonçalves, 2014 , https://portal.opengeospatial.org/files/?artifact_id=57477

OGC 14-055 OWS Context GeoJson Encoding, Pedro Gonçalves, 2015, Soon publicly available for public comments.

OGC 12-093 OWS-9 SSI UGAS Conversion Engineering Report.

Please check the Bibliography at the end of this document for additional references.

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard [OGC 06-121r9] shall apply. In addition, the following terms and definitions apply.

3.1
array
one of the data types that the value of a JSON key can have. I contains a sorted list of unnamed values

NOTE In fact, in JavaScript, an array is an object that has keys with consecutive numerical names.

3.2
declaration
associate an JavaScript object with a name of a data type.

3.3
define
Describe an JavaScript object data type by providing a list of its key properties, data type declarations and multiplicities.

NOTE As you will read later in the text JSON-LD is able to declare but not to define.

3.4
key
a JSON text that will represents the name of a variable in the JavaScript Document Object Model.

3.5
object
one of the data types that the value of a JSON key can have. It contains a list of property keys.

4 Conventions

4.1 Abbreviated terms

Some more frequently used abbreviated terms:

AJAX	Asynchronous JavaScript And XML
API	Application Program Interface

DOM	Document Object Model
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
OWL	Web Ontology Language
RDF	Resource Description Framework
WKT	Well Known Text
XML	Extendable Markup Language

4.2 UML notation

Diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 06-121r3].

5 JSON overview

This ER Topic addresses JSON and GeoJSON in OGC standards.

JavaScript Object Notation (JSON) is an open standard format that uses human-readable text but also a machine readable encoding to transmit data objects consisting of attribute–value (or arrays of values) pairs. The attribute is a quoted text and the values can be a quoted text, a number, or the words *true*, *false* and *null*. JSON is used primarily to transmit data between a server and web application, as an alternative for XML.

Although originally derived as a subset of JavaScript scripting language, JSON is a language-independent data format. Code for parsing and generating JSON data is available in many programming languages such as C++ or Java.

JSON is currently described by RFC 7159 and ECMA-404. The ECMA standard is minimal, describing only the allowed grammar syntax, whereas the RFC also provides some semantic and security considerations. The official Internet media type for JSON is application/json. The common JSON filename extension is ".json" but specific applications of JSON usually recommends other file extensions that contains the word "json" such us ".geojson".

This is an example of a river described in JSON:

```
{
   "river":
   {
      "name": "mississipi",
      "length": 3734,
      "discharge": 16790,
      "source": "Lake Itasca",
      "mouth": "Gulf of Mexico",
```



```
      "country": "United States of America",
      "bridges": ["Eads Bridge", "Chain of Rocks Bridge"]
   }
}
```

(http://en.wikipedia.org/wiki/Mississippi_River)

NOTE: One of the most annoying properties of the JSON encoding is that it is not possible to include comments in the file. This makes explaining the content inline impossible and commenting JSON fragments in this document more difficult.

This object is equivalent to this other one encoded in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<River>
    <name>mississipi</name>
    <length>3734</length>
    <discharge>16790</discharge>
    <source>Lake Itasca</source>
    <mouth>Gulf of Mexico</mouth>
    <country>United States of America</country>
    <bridge>Eads Bridge</bridge>
    <bridge>Chain of Rocks Bridge</bridge>
</River>
```

For AJAX applications, JSON is faster and easier to integrate in JavaScript code than XML. See the needed steps in the following comparison table:

Table 1: Steps to load values in JavaScript

Using XML you should do:	Using JSON you should do:
1. Fetch an XML document 2. Use the XML DOM to loop through the document 3. Extract values and store in variables	1. Fetch a JSON string 2. Parse the JSON string with JSON.parse(*string*)

Step 1: To fetch the JSON file in a JavaScript page you use the same function as to get a XML file: XMLHttpRequest

Step 2: Once you have the text stream, to load a “river” object in JavaScript you just need to do this:

```
var River = JSON.parse(text);
```

Then you can access any attribute in the object tree like any other variable structure. E.g. to access the river name you just do:

```
River.name;
```

The function JSON.parse() also validates the JSON stream syntactically and generates parse errors indicating any known problems. JSON.parse will parse only data ignoring methods of function definitions.

NOTE: First implementations of JSON parsing used the JavaScript eval() function. This is very risky since eval() will just not process data but any reference to methods of functions. This exposes a program to errant or malicious scripts. This is a serious issue when dealing with data retrieved from other Internet sites. The use of eval() is strongly discouraged.

In our opinion, JSON allows for the same things that can be done in XML (or a few less) but JSON's simplicity to immediately handle all attributes and embedded objects has made JSON popular among developers resulting in more agile development environments.

5.1 JSON an encoding without complex data types.

JSON inherits the flexibility of JavaScript. JavaScript is a language without class definition capability. This means that there are no complex data types; only "original" objects that are not associated to any predefined key (e.g. property) name list (i.e. each object has its own identity). Objects are created without associating them a class and properties are added during the creation or later when needed (at run time). This is a fundamental distinction with most of the common Object Oriented Languages and with GML. In fact, this approach is not new: in "simple" XML you can write an object name and add internal elements to it. The Document Object Model (DOM) will load them without associating them to any complex data type names. We can say that the document is syntactically valid. This is a powerful characteristic but prevents interoperability due to users receiving one of these files not knowing what to expect and the application is only able to present the data tree to the user for them to decide. Standards like ISO 19109 were created to provide a method to add geospatial data types (i.e. an application schema) and to increase interoperability by reducing freedom. GML follows ISO 19109 and adds geospatial data types to XML by using a XML mechanism for "autocontrol" the format: semantic validation. Semantic validation uses namespaces and XML schema files to define complex data types.

The fundamental question is: Do we what an equivalent mechanism to reduce the freedom in JSON objects and to be able to control complex data types? (simple data types are already better considered in RFC 7159 as will be mentioned later)They see at least 3 ways of doing this:

- A natural way is to mimic namespaces and schemas for JSON.
- A more indirect way is to link JSON to Linked data types and vocabularies like OWL or SKOS.

- Another alternative approach is to define the classes in UML and map them to JSON-LD data types declarations.

The next subclauses in this ER explore these three possibilities. Please note that GeoJSON already defines classes in a document (another less structured alternative) even if it does not provide an structured way of doing this such as schemas or UML. There is a long tradition to impose data models to geospatial information. For example, the INSPIRE directive for the creation of European SDI provides a list of datasets that need to be available at the European level classified in three Annexes. For each dataset (e.g Cadastral parcels from Annex I, Land Cover from Annex II, and Energy resources from Annex III) a data model encoded in UML, Feature Catalogue and GML has been provided and agreed to in a thematic working group. Being able to define classes (and complex data types) in a JSON encoding for features seems almost unavoidable.

5.2 JSON schema

A JSON schema is a document that defines the structure of JSON data. The Internet media type is "application/schema+json". JSON Schema defines what JSON keys are expected and the type of data values for a given application. JSON Schema is intended for semantic validation and documentation of data models. JSON schema acts in a similar way to XSD for a XML file. Indeed, some applications (such us XML Validator Buddy) are able to combine a JSON file with its corresponding JSON schema to test and validate if the content of the JSON file corresponds to the expected data model. Unfortunately, the level of control that JSON Schema provides is not as strict as XML Schema. The main problem lies in the fact that JSON objects are considered extendable by default. This means that adding attributes not specified in the schema does not give you an error. This prevents detecting object or attribute names with typos (that are confused with extended elements) except if they are declared as mandatory. Another difference is that JSON attributes are not supposed to have order so the order of the attributes of an object cannot be validated. In many cases this is not a problem since most of the data models used in the OGC do not depend on the other properties even if the XML “tradition” has imposed this unnecessary description.

NOTE: The fact that XML “sequence” imposes an order makes more complicated the validation of an Atom or a KML files (by design, both formats have their properties unsorted). This resulted in the use of RelaxNG and XSD 1.1 languages respectively for validating them (or to use the “choice” alternative to “sequence” as suggested by others).

If we suppose that all rivers share the same data model, the previous JSON instance example, can be validated against a JSON Schema like this:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "JSON minimal example",
    "description": "Schema for the minimal example that is a
river",
    "type": "object",
    "required": [ "River" ],
    "items": {
```

```
        "title": "Minimal River",
        "type": "object",
        "required": [ "name" ],
        "properties": {
            "name": {
                "type": "string"
            },
            "length": {
                "type": "number",
                "minimum": 0,
                "uom": "km"
            },
            "discharge": {
                "type": "number",
                "minimum": 0,
                "uom": "m^3/s"
            },
            "source": {
                "type": "string"
            },
            "mouth": {
                "type": "string"
            },
            "country": {
                "type": "string"
            },
            "bridges": {
                "type": "array",
                "items": {
                    "type": "string"
                }
            }
        }
    }
}
```

An interesting thing about JSON Schema is that it is also written in JSON (but using a predefined data model). This makes JSON Schema automatically extendable. We are using this property to extend the properties describing some of the attributes of the river already adding in the example a key "uom" that allows us to specify the units of measure of the numeric values of the "length" and "discharge" attributes. This could be useful for better defining feature types.

Recommendation 1: Consider extending JSON schema to fully describe the properties of a feature type, including units in alphanumeric properties and CRS in the geometric attributes instead of having to repeat them in each instance.
Target: OWS Common

Unfortunatelly, JSON schema is a IETF draft that expired in August 2013 and the future of the specification is uncertain. Fortunatelly there is still some activity in the blogs associated with the project even if one of the authors has recently blogged that he was forced to abandon the project due to lack of time.

Recommendation 2: Consider the possibility that OGC assists the IETF team in moving the JSON Schema forward.
Target: Architecture.DWG and OWS Common

Recommendation 3: Consider the possibility that OGC defines specific types for OGC/SIO geometry types.
Target: Architecture.DWG and OWS Common

NOTE: Most of the examples provided here have been validated syntactically and semantically using a windows application XML Validator Buddy. I appreciate the discussions with the developers of the product and some fast bug fixing or even fast adding of new functionalities.

5.3 JSON-LD

JSON-LD is a lightweight format initially designed for Linked Data (this is the “why” in the LD acronym). JSON-LD is based on JSON and provides a way to help JSON data to interoperate at Web-scale. The goal was to require as little effort as possible from developers to transform their existing JSON to JSON-LD. By defining the concept of a "@context" provides additional mappings from JSON to an RDF model. In practice, since there is an automatic way to go from JSON-LD to an RDF encoding, JSON-LD is considered also an encoding for RDF (a proof of this is that many programs that deal with RDF accept JSON-LD as input formant in addition to RDF/XML, turtle, n3, nq, etc. The “@context” links object properties in a JSON document to concepts in an ontology. A “@context” can be embedded directly in a JSON-LD document or written into a separate file and then referenced it from several other JSON files.

In practice JSON-LD can serve other purposes. JSON-LD can also help in the validation of a document. The reason is that it connects JSON variables to their definition in the semantic world (using “@id”) but also declaring data types (using “@type”). By connecting to a reestablished ontology, indirectly adds the idea of a namespace. This way, classes and its corresponding properties are defined by the ontology.

NOTE: JSON does not provide a mechanism for namespaces. There are some efforts to include them but some developers are worried to mimic XML too much reintroducing the complexity that JSON is avoiding.

Unfortunately, JSON-LD does not provide any object definition (a way to define which properties correspond to which objects), because this is supposed to be provided by the ontology itself. This means that objects and properties are defined in a “flat” list.

The following example incorporates “@context” to the previous JSON river example. Note the capacity to define each object and property using a URI and to define the type of some of the objects (in the absence of the “@type”, a string data type is supposed).

```
{
   "@context": {
```

```
      "xsd": "http://www.w3.org/2001/XMLSchema#",
      "uom": "http://www.opengis.net/def/uom",
      "River": "http://www.opengis.uab.cat/River",
      "name": "http://schema.org/name",
      "length": {
         "@id": "http://schema.org/distance",
         "@type": "xsd:float",
         "uom": "km"
      },
      "discharge": {
         "@id": "http://www.opengis.uab.cat/river/discharge",
         "@type": "xsd:float",
         "uom": "m^3/s"
      },
      "source": "http://www.opengis.uab.cat/riverSource",
      "mouth": "http://www.opengis.uab.cat/riverMouth",
      "country": "http://schema.org/nationality",
      "bridges": "http://www.opengis.uab.cat/riverBridge"
   },
   "River":
   {
      "name": "mississipi",
      "length": 3734,
      "discharge": 16790,
      "source": "Lake Itasca",
      "mouth": "Gulf of Mexico",
      "country": "United States of America",
      "bridges": ["Eads Bridge", "Chain of Rocks Bridge"]
   }
}
```

Again, a part of using @id and @type, the property “uom” is included making use of the JSON-LD extensibility.

To be able to link this object in the linked data an “@id” for the object is needed and a “@type” can replace the name of the class “River”.

```
{
   "@context": {
      "xsd": "http://www.w3.org/2001/XMLSchema#",
      "uom": "http://www.opengis.net/def/uom",
      "riverType": "http://www.opengis.uab.cat/River",
      "name": "http://schema.org/name",
      "length": {
         "@id": "http://schema.org/distance",
         "@type": "xsd:float",
         "uom": "km"
      },
```



```
      "discharge": {
         "@id": "http://www.opengis.uab.cat/river/discharge",
         "@type": "xsd:float",
         "uom": "m^3/s"
      },
      "source": "http://www.opengis.uab.cat/riverSource",
      "mouth": "http://www.opengis.uab.cat/riverMouth",
      "country": "http://schema.org/nationality",
      "bridges": "http://www.opengis.uab.cat/riverBridge"
   },
   "@id": "http://en.wikipedia.org/wiki/Mississippi_River",
   "@type": "riverType",
   "name": "mississipi",
   "length": 3734,
   "discharge": 16790,
   "source": "Lake Itasca",
   "mouth": "Gulf of Mexico",
   "country": "United States of America",
   "bridges": ["Eads Bridge", "Chain of Rocks Bridge"]
}
```

Now it is possible to automatically translate this into RDF in the *nquad* notation (http://www.w3.org/TR/n-quads/). One of the tools that executes this transformation is the JSON-LD playground (http://json-ld.org/playground/index.html)

```
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://schema.org/distance>
         "3734"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://schema.org/name>
         "mississipi" .
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://schema.org/nationality>
         "United States of America" .
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://www.opengis.uab.cat/river/discharge>
         "16790"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://www.opengis.uab.cat/riverBridge>
         "Chain of Rocks Bridge" .
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://www.opengis.uab.cat/riverBridge>
         "Eads Bridge" .
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://www.opengis.uab.cat/riverMouth>
         "Gulf of Mexico" .
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://www.opengis.uab.cat/riverSource>
         "Lake Itasca" .
```

```
<http://en.wikipedia.org/wiki/Mississippi_River>
         <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
         <http://www.opengis.uab.cat/River> .
```

NOTE: Similar attempts to transform a generic XML into RDF were also developed such as the Gleaning Resource Descriptions from Dialects of Languages (GRDDL).

JSON-LD was defined for making JSON to RDF conversions as easy as possible. Nevertheless, JSON-LD can be used for 2 more purposes:

- Defining namespaces
- JSON Validation

5.3.1 Using JSON-LD to define namespaces

JSON was defined with simplicity in mind. It is not considered good practice to introduce namespaces directly into key names. One of the reasons for not doing so is that the use of the common delimiter link ":" result in invalid JavaScript key names that can only be accessed with the more "baroque" notation. Let's consider the following JSON fragment:

```
wfs=JSON.parse('{"wfs:ServiceIdentification":{"ows:Title": "The
title"}}');
```

Accessing the title cannot be done by using normal "." Notation.

~~`wfs.wfs:ServiceIdentification.ows:Title`~~

Fortunately, JavaScript considers arrays and objects identical, so we can use the array notation to access the key:

```
wfs[wfs:ServiceIdentification][ows:Title]
```

JSON-LD allows for including a @context section enumerating the abbreviated namespace next to the URI namespace. Then the @context can contain different object and property names and the corresponding abbreviated namespace next to the name in that namespace.

```
{
   "@context":
   {
      "wfs": "http://www.opengis.net/wfs/2.5/",
      "ows": "http://www.opengis.net/ows/2.0/",

      "ServiceIdentification": "wfs:ServiceIdentification",
      "Title ": "ows:Title",
   }
```



```
   "ServiceIdentification":{
      "Title": "The title"
   }
}
```

By doing so, all JSON elements that are associated to a namespace URI can be dereferenced into a full URL when transformed to other RDF encoding such as *nquads*. Nquads examples in this document illustrate this mechanism.

5.3.2 Using JSON-LD to declare simple and complex types.

In JSON-LD we can declare the data type of all keys in a JSON file. When type is not declared a string type is assumed. This is valid for simple types (generality declared by using the "http://www.w3.org/2001/XMLSchema#" (normally abbreviated as “xsd”).

```
{
   "@context": {
      "xsd": "http://www.w3.org/2001/XMLSchema#",
      "discharge": {
         "@id": "http://www.opengis.uab.cat/river/discharge",
         "@type": "xsd:float",
      }
   }
   "discharge": 3.14
}
```

On the other hand complex types that can be defined by any other namespace (e.g.: http://schema.org”). You can declare that an object is of a complex type by adding a @type property (or a synonymous of “@type”) to it.

```
{
   "@context": {
      "geojson": "http://ld.geojson.org/vocab#",
      "type": "@type",
   }
   "geometry": {
      "type": "geojson:LineString",
      "coordinates": [
         [-95.2075, 47.239722], [-89.253333, 29.151111]
      ]
   }
}
```

The way complex data types are defined is out of scope of the JSON-LD specification that relies on the RDF way of defining complex types and vocabularies in OWL or SKOS.

On the other hand, JSON schema do allow for complex types definition:

```
{
   "type": "object",
   "required": [ "type", "id",  "properties"],
   "properties": {
      "id" : { "type": "string", "format": "uri" },
      "type": { "enum": [ "Feature" ] },
      "geometry": { "$ref": "#/definitions/geometry" },
      "properties": {
         "type": "object"
      }
   }
}
```

5.4 XML Schema, Schematron, JSON Schema, JSON-LD validation

The creation of a @context section in a JSON-LD introduces many elements that look similar to the ones introduced in JSON Schema. It seems reasonable to suppose that JSON-LD could be used by a validating algorithm to validate a JSON file in a similar way that JSON Schema does. The table 2 summarizes the capabilities provided by different validation strategies.

Table 2: Comparison of different validation approaches

Validation functionality	XML Schema	Schematron	JSON Schema	JSON-LD
Data types	yes		limited[4]	yes
Limits in simple data types	yes	yes	yes	no
Declare object of complex types	yes	no	yes	yes
Define complex data types	yes		yes	no[1]
Mandatory properties (multiplicity one) in objects	yes		yes	no[1]
More than one multiplicity of properties in objects	yes		Will be arrays[3]	no controlled
Order of the properties in objects	yes	no	no	no
Object tree dependency	yes		yes	with id's

				and links
Links between objects	xlink		not yet clear	with @type:@id
Unknown properties	yes		no[2]	yes
Unknown objects	yes		no[2]	yes
Namespaces	yes	yes	no	yes (by definition all keys are full URIs)
Conditional rules	no	yes	no	no
Connection to RDF	no	no	no	yes

[1] Could be provided by the vocabulary pointed by the URIs

[2] JSON is considered more flexible and extensible so an unknown property is considered an extension and it is ignored.

[3] use “type”:”array”, "minItems" : *min*, "maxItems": *max:* *http://stackoverflow.com/questions/23141511/how-to-map-uml-composition-cardinality-to-json-schema*

[4] limited to the JSON data types: “string”, “number”, “object”, “array”...

The authors of this ER believe that JSON-LD could be used as a validation strategy with the adoption of some additional conventions. In fact, many examples in this document have been validated using the JSON-LD playground. It is out of scope of this ER to try to completely assess this possibility but the authors recommend doing additional testing in the future.

Recommendation 4: Consider the combined use of JSON schema and the @context section of a JSON-LD file (possibly in combination with the ontologies linked to it) as a means for validating a JSON file in the OGC. The next OGC Testbed could include a test on this approach as an activity.
Target: Testbed-12

Recommendation 5: Consider the possibilities of using the namespace URIs in @context section of a JSON-LD file as a means to connect to formal ontologies structured in OWL SKOS or other RDF encoding as a way to validate complex types in JSON files in the OGC. The next OGC Testbed could include a test on this approach as an activity.
Target: Testbed-12

NOTE: https://developers.google.com/structured-data/testing-tool that will be mentioned later already verifies complex data structures in files written in JSON-LD and a similar approach is suggested here.

5.5 GeoJSON

In 2008, a group of individuals including some OGC members formed a community project to define and published a JSON encoding for simple geometries and features. The result of this work is GeoJSON. GeoJSON is a format for encoding simple feature geographic data structures. A GeoJSON object may represent a geometry, a feature, or a collection of features. GeoJSON supports the following geometric types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection. Features in GeoJSON contain a geometry object and additional properties, and features are grouped in a feature collection. Version 1 (also refereed as the 2008 version) was released in 16 June 2008 and can be found in geojson.org. Later, (2014) the group submitted a draft into the IETF process. At the moment they have been very active releasing 4 draft versions, the last one in February 2015.

This is how the river example, looks like encoded in GeoJSON:

```
{
   "type": "Feature",
   "geometry": {
      "type": "LineString",
      "coordinates": [
         [-95.2075, 47.239722], [-89.253333, 29.151111]
      ]
   },
   "properties": {
      "url": "http://en.wikipedia.org/wiki/Mississippi_River",
      "name": "mississipi",
      "length": 3734,
      "discharge": 16790,
      "source": "Lake Itasca",
      "mouth": "Gulf of Mexico",
      "country": "United States of America",
      "bridges": ["Eads Bridge", "Chain of Rocks Bridge"]
   }
}
```

As you can see, an object Feature has 3 members “type” “geometry” and “properties”. GeoJSON mainly sets restrictions on the values of type and in the content of the geometry element (that mainly contains an n dimensional array of coordinates) (see the red parts above). GeoJSON does not impose any restriction on the members of the properties, so they can be numbers, texts or other objects (only limited by the JSON types themselves).

As stated in OGC 14-113 OGC JSON Position Statement, GeoJSON is getting momentum and “OGC members and the broader geo-community are using or intending on using JSON encodings for some or all of their applications that require geographic data encoding and transfer”. Many people are starting to distribute maps in the internet in

this format (e.g.: https://github.com/johan/world.geo.json) and applications to download (e.g.: http://geojson-maps.kyd.com.au/) and editing online (e.g.: http://geojson.io/) are proliferating. In particular, GitHub offers the possibility to upload GeoJSON files with versioning (https://help.github.com/articles/mapping-geojson-files-on-github).

5.5.1 Comparing conceptual limitations in GeoJSON and in GML-SF

GeoJSON separates the geometry from the alphanumeric properties. This is not done in GML. Indeed, GML features have an array of properties, some of them derived from GML geometric types and others derived from XML types. In the GeoJSON case, geometry can be as simple as a single point and as complex as a geometry collection. GeoJSON *geometry* has no semantics associated with it so there is no information about what the *geometry* is representing (the centre of the road, the margins, the asphalted zone...). To overcome the GeoJSON restriction of having a single *geometry*, a geometry collection can be used, but again, no semantics are associated with the elements of the collection. GML-SF does not impose restrictions on the number of geometric properties and in theory can have several geometries per feature and the properties that will have names helping to identify the meaning. In practice instances of GML-SF rarely use more than one geometric property. A possible solution to overcome this problem is to include GeoJSON geometries in the *properties* array. In fact, nothing in the current GeoJSON standard prevents the use of other geometrical descriptions in the properties array even if current parsers probably will not recognize them.

GML-SF0 and GML-SF1 can only have simple non-geometrical properties (such strings of numbers) explicitly excluding complex structures. GeoJSON imposes no restriction on its properties (and the can be strings, numbers or objects). In this sense, GeoJSON is at the same level as GML-SF2.

GML-SF can define a model that limits the type of the geospatial property to single defined type (e.g.: a point). GeoJSON does not impose any homogeneity rules on the geometrical properties. This way, a feature collection can have features with a mixture of geometric points, lines, polygons, etc. GeoJSON does not impose any restrictions on the non-geometric properties either. If necessary some restrictions can be imposed by using a specific JSON schema as will be discussed later.

5.5.2 Comparing GeoJSON coordinates with WKT

The following table compares the JSON coordinates notation with the Well Known Text (WKT) notation (WKT is defined in OGC 06-103r4, OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture). This will be helpful both to determine which subset of WKT GeoJSON covers and also to see parallelism between both encodings

Table 3: Comparing JSON coordinates with WKT notation

Geometry	Text Literal	Coordinates JSON	Comment

Type	Representation		
Point	`Point (10 10)`	`"type": "Point", "coordinates": [10, 10]`	a Point
LineString	`LineString ( 10 10, 20 20, 30 40)`	`"type": "LineString", "coordinates": [[10, 10], [20, 20], [30, 40]]`	a LineString with 3 points
Polygon	`Polygon ((10 10, 10 20, 20 20, 20 15, 10 10))`	`"type": "Polygon", "coordinates": [[[10, 10], [10, 20], [20, 20], [20, 15], [10,10]]]`	a Polygon with 1 exteriorRing and 0 interiorRings
Multipoint	`MultiPoint ((10 10), (20 20))`	`"type": "MultiPoint", "coordinates": [[10, 10], [20, 20]]`	a MultiPoint with 2 points
MultiLine String	`MultiLineString((10 10, 20 20), (15 15, 30 15))`	`"type": "MultiLineString", "coordinates": [[[10, 10], [10, 20]], [[15, 15], [30, 15]]]`	a MultiLineString with 2 linestrings
MultiPoly gon	`MultiPolygon( ((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60 )) )`	`"type": "MultiPolygon", "coordinates": [[[[10, 10], [10, 20], [20, 20] , [20, 15], [10, 10]]], [[[60, 60], [70, 70], [80, 60], [60, 60]]]]`	a MultiPolygon with 2 polygons
GeomColl ection	`GeometryCollection ( POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20) )`	`"type": "GeometryCollection", [{ "type": "Point", "coordinates": [10, 10] },{ "type": "Point", "coordinates": [30, 30]] },{ "type": "LineString", "coordinates": [[15, 15], [20, 20]]`	a GeometryCollec tion consisting of 2 Point values and a LineString value

		}]	
Polyhedron	Polyhedron Z (((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1. 0 0 1)))	N/A	A polyhedron cube, corner at the origin and opposite corner at (1, 1, 1).
Tin	Tin Z (((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 0 0 1, 0 0 0)), ((1 0 0, 0 1 0, 0 0 1, 1 0 0)),)	N/A	A tetrahedron (4 triangular faces), corner at the origin and each unit coordinate digit.
Point	Point Z (10 10 5)	"type": "Point", "coordinates": [10, 10, 5]	a 3D Point
Point	Point ZM (10 10 5 40)	N/A	the same 3D Point with M value of 40
Point	Point M (10 10 40)	N/A	a 2D Point with M value of 40

5.6 TopoJSON

TopoJSON is an extension of GeoJSON that encodes only a specific topological case: 2D planar topological polygons composed by sequences of edges. Rather than representing geometries directly as arrays of coordinates, polygons geometries are defined as sequences of edges (actually TopoJSON call them “arcs”). Each edge is defined only once, but can be referenced several times by different shapes, thus reducing redundancy and decreasing the file size and it is specified here: https://github.com/topojson/topojson-specification/blob/master/README.md.

Some state that a typical TopoJSON file is 80% smaller than its GeoJSON equivalent. A JavaScript library such as https://github.com/mbostock/topojson can be used to transform form TopoJSON to GeoJSON.

The last river document could be presented in TopoJSON as:

```
{
```

```
   "type":"Topology",
   "transform":{
     "scale": [1,1],
     "translate": [0,0]
   },
   "objects":{
     "mississipi_river":{
       "type":"GeometryCollection",
       "geometries":[
        {
      "type": "LineString",
      "arcs": [0],
      "properties":{
         "url": "http://en.wikipedia.org/wiki/Mississippi_River",
         "name": "mississipi",
         "length": 3734,
         "discharge": 16790,
         "source": "Lake Itasca",
         "mouth": "Gulf of Mexico",
         "country": "United States of America",
         "bridges": ["Eads Bridge", "Chain of Rocks Bridge"]
      }
   }
        ]
     }
   },
   "arcs": [[[-95.2075, 47.239722], [-89.253333, 29.151111]]]
}
```


In particular, a JavaScript code has been developed to transform TopoJSON in GeoJSON. This way, JavaScript clients supporting GeoJSON can automatically support TopoJSON.

The work on TopoJSON is out of scope of this ER but it is showing us an accepted way of extending GeoJSON into other paradigms that can support characteristics that the core GeoJSON does not cover. The possibility of having a JavaScript transformation code that is able to convert a JSON file into a valid GeoJSON file is worth considering as a way to move forward.

Recommendation 6: Consider TopoJSON as a model to create a JSON encoding that is different (not just an extension, because addresses a topic that GeoJSON can not consider) but can be mapped and automatically converted into a GeoJSON file (using for example a JavaScript library).
Target: OWS Common

Recommendation 7: Connect work in previous testbeds about a WPS profile for topological applications with the TopoJSON to study the applicability and interoperability of TopoJSON in OGC standards such as WPS and WFS.
Target: Testbed 12

5.7 Symbology in GeoJSON

A very simple vendor specification has been produced to extend GeoJSON to include a minimum control of its symbology: https://github.com/mapbox/simplestyle-spec/tree/master/1.1.0. This specification defines a set of keys that can be included in the properties array of each feature to define some visualization properties. These properties are related to markers sizes and shapes, and colors for polygons and lines.

```
{
    "type": "FeatureCollection",
    "features": [{ "type": "Feature",
        "geometry": {
            "type": "Point",
            "coordinates": [0, 0]
        },
        "properties": {
            "description": "Bus stop",
            "marker-size": "medium",
            "marker-symbol": "bus",
            "marker-color": "#ace"
        }
    }, {
        "type": "Feature",
        "geometry": {
            "type": "LineString",
            "coordinates": [[0, 0], [10, 10]]
        },
        "properties": {
            "description": "Bus path",
            "stroke": "#f0f0f0",
            "stroke-width": 2
        }
    }]
}
```

5.8 Current status in OGC

Traditionally the OGC has used XML to encode data and service descriptions of any kind. Discussions as to the merits of JSON as a competing encoding or as an alternative to facilitate the implementation of OGC standards have been ongoing. As a result, the OGC 14-113 OGC JSON Position Statement was approved by the OGC members and released. Most of the information in this subclause has been extracted from that document.

Currently not many OGC standards are using JSON. Some standard authors are reluctant to propose standard ways of using JSON or GeoJSON without a clear guidance from the TC as a whole. One surprising example is that several OGC members developed commercial WFS servers that are able to respond with a GeoJSON payload (as well as other encodings) but the new version of WFS standard version 2.5 drafts still does not

describe clearly support for either JSON or GeoJSON (even if it provides some recommendations). The most concrete example of standard candidate using JSON or GeoJSON is:

- OGC 14-055, OWS Context GeoJSON Encoding Standard. Submitted to Pending August 2014. https://portal.opengeospatial.org/files/?artifact_id=59982&version=1
    - This candidate standard extends GeoJSON to include the elements coming from the OWS Context conceptual model. “The goal of this standard is to provide a definition of how to encode a context document, which can be extended to allow a context referencing a fully configured service set, which can be defined and consistently interpreted by clients”.

Works in previous testbeds is mentioned throughout this document: OGC 14-009r2, Testbed-10 Rules for JSON and GeoJSON Adoption: Focus on OWS-Context (March 2014) and OWS 12-093 UGAS Conversion Engineering Report

Some other document part of the OGC process has been uploaded to the OGC systems:

- [OWS Common] Define XML and JSON schema for a web linking structure based on RFC 5988 (Change Request)
- XACML 3.0 JSON Profile (Presentation)

6 Deriving a JSON encoding from XML and UML

JSON may be an alternative to XML, providing better integration with other standards making OGC standard implementation more accessible. Even if JSON does not provide useful technologies such as XSLT or namespaces, the possibility of including JSON-LD in the JSON encodings opens a door for fully integrating two ways of describing entities: object/features and RDF/semantics in a single encoding.

6.1 Derive JSON from XML

As stated, most of the OGC encoding and services rely on XML. Therefore, it would be interesting to provide a way to directly transform XML encodings into JSON encodings. This was already explored in the previous Testbed 10 in the Engineering Report: “OGC 14-009r1, OGC Testbed-10 Rules for JSON and GeoJSON Adoption: Focus on OWS-Context” and we are mainly adopting this work here but with 2 exceptions to the general rules.

6.1.1 General Rules for transforming XML into JSON

This is the summary of the general transformation rules in OGC 14-009r1:

- The XML element local name is the JSON object name.
- The XML element single text node is the JSON object value.
- The XML element attributes nodes are transformed in JSON nested objects (see OGC 14-009r1 section 6.1.1).
- The XML nested elements are transformed in JSON nested objects (see OGC 14-009r1 section 6.1.2).
- A XML element text node is transformed in a JSON nested object when other types of nodes are present (see OGC 14-009r1 section 6.1.3).
- The XML element text value can be casted to a JSON object value type (see OGC 14-009r1 section 6.2).
- XML fragments can be transformed in text members (see OGC 14-009r1 section 6.3).

In addition, the following two recommendations are not in the original list but the text of the document suggest they are as important as the previous ones.

- XML repeated elements (or attributes) are transformed into JSON arrays. The JSON array name could be changed to plural when convenient (see OGC 14-009r1 section 6.1.4).
- XML namespaces are ignored.

6.1.1.1 The rule of plural

The plural rule in OGC 14-009r1 seems reasonable but we have to allow for some exceptions. For example, some current XML documents in OGC (e.g. a WFS ServiceMetadata document) will generate unnatural translations:

```
<ows:Keywords>
   <ows:Keyword>WFS</ows:Keyword>
   <ows:Keyword>WMS</ows:Keyword>
   <ows:Keyword>GEOSERVER</ows:Keyword>
</ows:Keywords>
```

Translation into JSON:

```
"Keywords":
{
   "Keywords": [ "WFS", "WMS", "GEOSERVER" ]
}
```

results in unnecessary repetition of the “Keywords” word that can probably be simplified.

Please note that some XML elements can already be plural resulting in JSON plural names even if they are not arrays. The fact that the name is plural cannot be used as a way to anticipate if a key is an array. In many senses objects and arrays are considered equivalent in JavaScript (an array is considered an object with numeric members) so identifying an array is not completely straightforward. Proof of this is that the JavaScript operator “typeof” returns “object” both for “objects” and “arrays”.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
typeof fruits;             // typeof returns object
```

To solve this problem you can create your own isArray() function [1]:

```
function isArray(myArray) {
    return myArray.constructor.toString().indexOf("Array") > -1;
}
```

6.1.1.2 Mixed elements

OGC 14-009r1 Section 6.1.3 states: “For this XML mixed element the text node will be transformed into a JSON object where the name is the parent XML element name and the value is the text node contents.”

For example, the following XML fragment:

```
<branch olive="true">
not empty
<leaf>green</leaf>
<peach type="red">some</peach>
</branch>
```

results in:

```
"branch" : {
   "branch" : "not empty",
   "olive" : "true",
   "leaf" : "green",
   "peach" : {
      "peach" : "some",
      "type" : "red"
   }
}
```

[1] http://www.w3schools.com/js/js_arrays.asp

NOTE: the line '"peach" : "some",' is missing in the original example.

To access the text node you simple repeated the name of the node: "branch.branch"

The section 6.1.3 in OGC 14-009r1 does not state that a combination of a XML element containing a text node and one of more attributes is in the scope of the same problem even if these cases cannot be called a mixed XML element. This case is much more common in the OGC. For example, this situation is already present in WFS Capabilities documents such as this one: (http://cida.usgs.gov/nwc/geoserver/NHDPlusFlowlines/ows?service=WFS&version=1.0.0&request=GetCapabilities):

```
<ogc:Function_Name nArgs="1">abs</ogc:Function_Name>
```

that will end in:

```
"ogc_Function_Name":
{
    "ogc_Function_Name": "abs";
    "nArgs":"1"
}
```

6.1.1.3 NULL elements

OGC 14-009r1 Section 6.2.4 says: “The XML empty elements must be explicitly transformed to the null JSON object.”

This way an XML fragment like this:

```
<tree value="false">
<child/>
</tree>
```

is transformed into this:

```
"tree" : {
   "child" : null
}
```

In this explanation, it would be good to add this similar case:

```
<Get
onlineResource="http://cida.usgs.gov:80/nwc/geoserver/NHDPlusFlow
lines/wfs?request=GetCapabilities"/>
```

In this case there is no need to generate a null element and the transformation would be:

```
"Get":
```

```
{
   "onlineResource":
"http://cida.usgs.gov:80/nwc/geoserver/NHDPlusFlowlines/wfs?reque
st=GetCapabilities"
}
```

In addition, it could be useful to differentiate an empty attribute from a null element. For example:

```
<tree value="false">
<child/>
<child2></child2>
</tree>
```

it will be transformed into this:

```
"tree" : {
   "child" : null
   "child2" : ""
}
```

Recommendation 8: Produce an OGC best practice for converting XML documents into JSON based on OGC 14-009r1 and some other considerations exposed in this ER.
Target: OWS Common

6.1.2 Exceptions to the general rules

6.1.2.1 Encoding the Object-property alternation in JSON

GML and ISO 19115 are two examples of documents that use the object-property model where objects names (in fact the class names) in UpperCamelCase contain only property names in lowerCamelCase. Properties can be defined as objects (again in UpperCamelCase). When translating into JSON the class name needs to be removed and substituted by a *type* key (e.g. "@type") with a reference to a class type name.

This way, the following XML fragment:

```
<mdb:MD_Metadata>
   <mdb:contact>
      <cit:CI_Responsibility>
         <cit:party>
            <cit:CI_Organisation>
               <cit:name>
                  <gco:CharacterString>Institut CartogrÃ fic de
Catalunya (ICC)</gco:CharacterString>
               </cit:name>
               <cit:contactInfo>
                  <cit:CI_Contact>
```



```
                        <cit:address>
                           <cit:CI_Address>
                              <cit:deliveryPoint>
                                 <gco:CharacterString>Parc de
MontjuÃ¯c</gco:CharacterString>
                              </cit:deliveryPoint>
                              <cit:city>

   <gco:CharacterString>Barcelona</gco:CharacterString>
                              </cit:city>
                              <cit:postalCode>
                                 <gco:CharacterString>E-
08038</gco:CharacterString>
                              </cit:postalCode>
                              <cit:electronicMailAddress>

   <gco:CharacterString>centre.atencio@icc.cat</gco:CharacterStri
ng>
                              </cit:electronicMailAddress>
                           </cit:CI_Address>
                        </cit:address>
                     </cit:CI_Contact>
                  </cit:contactInfo>
               </cit:CI_Organisation>
            </cit:party>
         </cit:CI_Responsibility>
   </mdb:contact>
</mdb:MD_Metadata>
```

should be encoded in JSON like this:

```
{
   "@type": "mdb:MD_Metadata",
   "contact":{
      "@type": "cit:CI_Responsibility",
      "party": {
         "@type": "cit:CI_Organisation",
         "name": "Institut Cartogràfic de Catalunya (ICC)",
         "contactInfo": {
            "@type": "cit:CI_Contact",
            "address": {
               "@type": "cit:CI_Address",
               "deliveryPoint": "Parc de Montjuïc",
               "city": "Barcelona",
               "postalCode": "E-08038",
               "electronicMailAddress": "centre.atencio@icc.cat"
            }
         }
      }
   }
```

```
}
```

NOTE: There is also a project that is targeting the encoding of ISO and FGDC metadata in JSON: https://github.com/adiwg

Recommendation 9: Include adding "@type" keys to JSON objects as a good practice to makethe transition to JSON-LD and RDF easier. It is also good practice that type names are qualified with a abbreviated namespaces (e.g.: ows:ServiceIdentification) that could be later dereferenced using JSON-LD @context.
Target: OWS Common with OAB

The GML case is even more special because GeoJSON should be considered and an alternative encoding for vector features. GeoJSON can be extended to fully support all geospatial types that GML support if needed. Nevertheless, OGC membership can perhaps select another approach that would be more suitable in the future

6.1.2.2 Linking in JSON

NOTE: This section is a response to the demand expressed in the CR-242 (OGC 12-121) requesting that OWS Common includes a recommendation for expressing links in JSON that can allow for a functionally similar to the one provided by XLink.

Many OGC standards use xlink (or other similar forms of links) to relate elements in the same document or in a remote document. A direct translation of these links will not result in the right conversion to JSON-LD. In a general case, there is a need for encoding the source of the link, a target url, the reason why this link is required, the title of the link, the MIME type recovered from the link and the expected size.

There are several encodings proposed in the Internet to do this in JSON but none of them are gaining more momentum than others. Some of them can be seen here:

- http://json-schema.org/links
- http://amundsen.com/media-types/collection/examples/
- https://github.com/kevinswiber/siren
- https://gist.github.com/miyagawa/1912431
- http://tools.ietf.org/html/draft-kelly-json-hal-06
- http://blog.cto.hiv/relations-in-linked-data/

It is difficult to find arguments in favor of a particular approach until a conversion into JSON-LD is explored. In JSON-LD all objects need to have a *key* that is considered a synonymous to “@id”. This *key* can be a target of any relation. A relation can be established in source object as an attribute with a key name stating the reason of the link that can be mapped with the *atom link rel* values. This key will also be a link object with extra properties. The object will have an key synonymous of “@id” is considered the target object and a key synonymous of “@type” indicating that this is a “link” object in the “atom” namespace. The object can have extra parameters about the MIMEtype and

the expected size of the object. For convenience the reason for the link can be encapsulated in a “link” key.

In the following example, we are relating (linking) the representation of the Rio Negro river in Wikipedia with the Amazon river and also to a metadata record better describing the Rio Negro.

```
{
   "type": "pg:River",
   "id": "wiki:Rio_Negro_(Amazon)",
   "tributes":
   {
      "href": "wiki:Amazon_River",
      "type": "pg:River",
   },
   "links":
   {
      "id": "wiki:Rio_Negro_(Amazon)",
      "via":
      {
         "type": "atom:link",
         "MIMEtype": "application/xml",
         "href": "http://www.river.com/MetadataRioNegro.xml",
         "title": "XML metadata for the Rio Negro river",
         "length": 1523,
      }
   }
}
```

Using the following context:

```
   "@context":
   {
      "atom": "http://www.w3.org/2005/Atom/",
      "wiki": "http://en.wikipedia.org/wiki/",
      "pg": "http://physicalgeography.schema.org/",

      "id": "@id",
      "href": "@id",
      "type": "@type",
      "MIMEtype": "atom:type",

      "links": "_:",
      "via": "atom:via",
      "title": "atom:title",
      "length": "atom:length",
      "tributes": "pg:tributes",
   }
```

We can automatically transform the JSON file into RDF where the relations between objects are highlighted in green.

```
<http://en.wikipedia.org/wiki/Rio_Negro_(Amazon)>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://physicalgeography.schema.org/River> .

<http://en.wikipedia.org/wiki/Rio_Negro_(Amazon)
      <http://physicalgeography.schema.org/tributes>
      <http://en.wikipedia.org/wiki/Amazon_River> .

<http://en.wikipedia.org/wiki/Amazon_River>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://physicalgeography.schema.org/River> .

<http://en.wikipedia.org/wiki/Rio_Negro_(Amazon)>
      <http://www.w3.org/2005/Atom/via>
      <http://www.river.com/MetadataRioNegro.xml> .

<http://www.river.com/MetadataRioNegro.xml>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.w3.org/2005/Atom/link> .
<http://www.river.com/MetadataRioNegro.xml>
      <http://www.w3.org/2005/Atom/length>
      "1523"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.river.com/MetadataRioNegro.xml>
      <http://www.w3.org/2005/Atom/title>
      "XML metadata for the Rio Negro river" .
<http://www.river.com/MetadataRioNegro.xml>
      <http://www.w3.org/2005/Atom/type>
      "application/xml" .
```

Recommendation 10: Include in a best practice for JSON a subclause for linking to other objects in JSON, using the natural approaches that JSON-LD provides for both simple links and atom links. Target: OWS Common.SWG

6.1.2.3 Geospatial objects

XML objects that can be described as simple features (in particular GML objects) and Bounding Boxes should be encoded in GeoJSON geometric elements.

For instance, the <ows:WGS84BoundingBox> object, should be converted into a bbox GeoJSON objects. For example, the following WFS ServiceMetadata fragment:

```
<FeatureTypeList>
    <FeatureType>
      <Name>NHDPlusFlowlines:PlusFlowlineVAA_NHDPlus18</Name>
      <Title>PlusFlowlineVAA</Title>
```



```
      <Abstract/>
      <DefaultCRS>urn:ogc:def:crs:EPSG::900913</DefaultCRS>
      <ows:WGS84BoundingBox>
        <ows:LowerCorner>-124.40958558399815
32.50005761536461</ows:LowerCorner>
        <ows:UpperCorner>-114.58848453257575
43.33627233179173</ows:UpperCorner>
      </ows:WGS84BoundingBox>
    </FeatureType>
   ...
```

Should be converted into:

```
{
   "Name": "NHDPlusFlowlines:FeatureTypeList",
   "FeatureType": [
   {
      "Name": "NHDPlusFlowlines:PlusFlowlineVAA_NHDPlus18",
      "Title": "PlusFlowlineVAA",
      "Abstract": null,
      "DefaultCRS": "urn:ogc:def:crs:EPSG::900913",
      "bbox": [-124.40958558399815 32.50005761536461,
             -114.58848453257575, 43.33627233179173]
   }
}
```

The corresponding context could be:

Using the following context:

```
   "@context":
   {
      "wfs": "http://www.opengis.net/wfs/2.5/",
      "ows": "http://www.opengis.net/ows/2.0/",
      "geojson": "http://ld.geojson.org/vocab#",

      "Name": "@id",
      "Title": "ows:Title",
      "Abstract": "ows:Abstact",
      "DefaultCRS": "wfs:DefaultCRS",
      "bbox": "geojson:bbox",
   }
```

Recommendation 11: Include in the JSON best practice that if a fragment of a XML document contains a geospatial object then when converting to JSON, consider using the GeoJSON equivalent type.
Target: OWS.Common

6.2 Derive JSON from UML

Deriving JSON form UML is out of scope of this Engineering Report. The Engineering Report elaborated in the Testbed 9 OGC 12-093 “OWS-9: UML-to-GML-Application-Schema (UGAS) Conversion Engineering Report”, elaborates on how to derive JSON encodings from UML automatically.

7 Discussion about GeoJSON

7.1 JSON Schema for GeoJSON

In the same way that a GML instance can be described by a GML application schema (XSD file) a GeoJSON file can be described by a JSON Schema. There are two approaches for this:

- A generic JSON schema describing any GeoJSON file.
- A specific JSON schema describing a particular GeoJSON file of a particular feature type.

7.1.1 Generic GeoJSON validation

https://github.com/fge/sample-json-schemas/tree/master/geojson provides an attempt to generate a GeoJSON generic JSON schema. This can be useful to validate that a JSON file is in fact a GeoJSON file but does not provide the “GML application schema” functionality.

We have done a similar exercise during the process of defining a JSON Schema for OWS Context as seen later.

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "GeoJSON schema",
   "type": "object",
   "required": [ "type" ],
   "properties": {
      "type": { "enum": [ "FeatureCollection" ] },
      "id" : { "type": "string", "format": "uri" },
      "bbox": {
         "type": "array",
         "items": {
            "type": "array",
            "minItems": 4,
            "items" : { "type": "number" }
         }
      },
      "features" : {
```



```
          "type": "array",
          "items": {
            "type": "object",
            "required": [ "type", "properties"],
            "properties": {
               "type": { "enum": [ "Feature" ] },
               "geometry": { "$ref": "#/definitions/geometry" },
               "properties": { "type": "object" }
            }
          }
      }
   },
   "definitions": {
      "geometry": {
         "title": "geometry",
         "type": "object",
         "oneOf": [{
            "properties": {
               "type": { "enum": [ "Point"] },
               "coordinates": {
                  "type": "array",
                  "minItems": 2,
                  "items": { "type": "number"}
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "LineString", "Multipoint"]},
               "coordinates": {
                  "type": "array",
                  "minItems": 2,
                  "items": {
                     "type": "array",
                     "minItems": 2,
                     "items" : { "type": "number" }
                  }
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "Polygon",
"MultiLineString"]},
               "coordinates": {
                  "type": "array",
                  "items": {
                     "type": "array",
                     "minItems": 2,
                     "items": {
                        "type": "array",
```

```
                        "minItems": 2,
                        "items" : { "type": "number" }
                     }
                  }
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "MultiPolygon"] },
               "coordinates": {
                  "type": "array",
                  "items": {
                     "type": "array",
                     "items": {
                        "type": "array",
                        "minItems": 2,
                        "items": {
                           "type": "array",
                           "minItems": 2,
                           "items" : { "type": "number" }
                        }
                     }
                  }
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "GeometryCollection"]},
               "geometries": {
                  "type": "array",
                  "items": { "$ref": "#/definitions/geometry" }
               }
            }
         }]
      }
   }
}
```

7.1.2 Specific GeoJSON validation

The generic schema presented before leaves open the nature of the *geometry* and the content of *properties*. A specific JSON schema that acts like a GML application schema could be more appropriate to better describe a GeoJSON instance type. In it we can limit:

- The possibilities of geometry to a specific geometry type (e.g. LineString)

- The content of properties to a set of recognized list of attributes (even if in JSON schema we cannot validate the inclusion of unknown properties).

This is an example of a river collection JSON file:

```
{
   "id": "river:ExampleRiverCollection",
   "type": "FeatureCollection",
   "features":
   [
      {
         "id": "wikipedia:Mississippi_River",
         "type": "Feature",
         "geometry":
         {
            "id": "wikipedia:Mississippi_River",
            "type": "LineString",
            "crs": "ogc_def:crs/OGC/1.3/CRS84",
            "coordinates":
            [
               [
                  -95.2075,
                  47.239722
               ],

               [
                  -89.253333,
                  29.151111
               ]
            ]
         },
         "properties":
         {
            "url": "wikipedia:Mississippi_River",
            "type": "riverType",
            "name": "mississipi",
            "length": 3734,
            "discharge": 16790,
            "source": "Lake Itasca ",
            "mouth": "Gulf of Mexico",
            "country": "United States of America",
            "bridges":
            [
               "Eads Bridge",
               "Chain of Rocks Bridge"
            ]
         }
      },

      {
         "id": "wikipedia:Ebro",
```

```
         "type": "Feature",
         "geometry":
         {
            "id": "wikipedia:Ebro",
            "type": "LineString",
            "crs": "ogc_def:crs/OGC/1.3/CRS84",
            "coordinates":
            [
               [
                  -4.402942,
                  43.039111
               ],

               [
                  0.863056,
                  40.72
               ]
            ]
         },
         "properties":
         {
            "url": "wikipedia:Ebro",
            "type": "riverType",
            "name": "ebro",
            "length": 930,
            "discharge": 426,
            "source": "Pico Tres Mares",
            "mouth": "Mediterranean Sea",
            "country": "Spain"
         }
      }
   ]
}
```

This is how a JSON schema for validating this featureType looks:

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "Specific GeoJSON schema for riverType",
   "type": "object",
   "required": ["type"],
   "properties":
   {
      "type": {
         "enum": [ "FeatureCollection"]
      },
      "id": {
         "type": "string",
```



```
            "format": "uri"
         },
         "bbox": {
            "type": "array",
            "items": {
               "type": "array",
               "minItems": 4,
               "items": {
                  "type": "number"
               }
            }
         },
         "features": {
            "type": "array",
            "items": {
               "type": "object",
               "required": [ "type", "geometry","properties"],
               "properties":
               {
                  "type": {"enum": ["Feature"]},
                  "geometry": {
                     "$ref": "#/definitions/geometry"
                  },
                  "properties": {
                     "type": "object",
                     "required": ["type", "name", "country"],
                     "properties":
                     {
                        "url": { "type": "string", "format": "uri"
},
                        "type": { "enum":  ["riverType"] },
                        "name": { "type": "string"},
                        "length": { "type": "number"},
                        "discharge": { "type": "number"},
                        "source": { "type": "string"},
                        "mouth": { "type": "string"},
                        "country": { "type": "string"},
                        "bridges": {
                           "type": "array",
                           "items": {
                              "type": "string"
                           }
                        }
                     }
                  }
               }
            }
         }
      },
      "definitions":
      {
```

```
        "geometry":
        {
            "title": "geometry",
            "type": "object",
            "properties":
            {
                "type":
                {
                    "enum":
                    [
                        "LineString"
                    ]
                },
                "coordinates":
                {
                    "type": "array",
                    "minItems": 2,
                    "items":
                    {
                        "type": "array",
                        "minItems": 2,
                        "items":
                        {
                            "type": "number"
                        }
                    }
                }
            }
        }
    }
}
```

Recommendation 12: Adopt the creation of specific JSON schema documents as a means of defining feature types and as a means for feature instance validation (as the equivalent of GML application schema).
Target: WFS and OAB

NOTE: the adoption of this strategy will give GeoJSON implementations more robustness. It is particularly important to increase interoperability of the services that has to process data.

7.2 GeoJSON in JSON-LD

JSON-LD provides a transformation *template* called "@context" that indirectly defines a set of rules or transforming a JSON file into an RDF. This is similar to writing a XSLT but using a complete different language.

In this section we describe how to convert the previous GeoJSON example into a useful JSON-LD. To do so, there are 3 main issues:

- Create the right “@context” code to give semantics to the “key names”.
- Remove the “properties” node.
- Express the coordinates correctly in RDF.

To solve the first issue, we use a subset of the experimental vocabulary that is found in http://geojson.org/vocab that sometimes uses the URI http://example.com/vocab# and some other the http://ld.geojson.org/vocab# that is supposed to contain the GeoJSON concepts. In fact the complete vocabulary can be found at http://geojson.org/contexts/geojson-base.jsonld. Our subset contains geojson:coordinates, geojson:Feature and geojson:LineString. We also used the URI http://www.opengis.uab.cat/River/ to define the semantics of the river properties

To solve the second I had to apply a little “trick” using the same “id” for the root object, the geometry object and the properties object and associate “geometry” and “properties” to the “_:”void uri.

GeoJSON is very clear about the encoding of the coordinates. Coordinates are n-dimensional arrays of numbers. Unfortunately sorted n-dimensional arrays cannot be exported to RDF. By default, in JSON-LD, the elements of an array are considered an “unsorted” list of values, so that they become unsorted in the RDF transformation. JSON-LD provides a methodology to transform a sorted array (using “@container”: “@list”), but, unfortunately the implementations we have tested, only make it possible for one dimensional array. In fact, this issue is discussed in https://github.com/geojson/geojson-ld/issues/28 (and with less completeness in https://github.com/geojson/geojson-ld/issues/26 and https://github.com/geojson/geojson-ld/issues/12) that were closed with no solution. In conclusions, the current GeoJSON encoding for coordinates prevents the correct automatic transformation to RDF (via JSON-LD).

Recommendation 13: Consider carefully the unsolved issue where GeoJSON coordinates prevents a natural way to apply JSON-LD to GeoJSON and an automatic conversion to RDF. Following recommendations are proposing alternative solutions.
Target: OAB

It seems that a logical solution is to propose an encoding that is not based on arrays such us the one used here http://geovocab.org/geometry.html or the Well Known Text (WKT) (http://www.opengeospatial.org/standards/sfa). WKT encoding can be applied in two methods:

- As a new encoding for JSON features that will break compatibility with GeoJSON.
- As an intermediate encoding for the JSON to RDF conversion. This way we will be able to convert any existing GeoJSON coordinates array during the JSON to RDF conversion.

7.3 The need for an alternative to GeoJSON. Another encoding for features in JSON.

7.3.1 Reason to define an GeoJSON alternative encoding

Limitations of the current version of GeoJSON are:

- Coordinates array cannot be converted to RDF using JSON-LD due to the fact that multidimensional arrays are very complicated to represent in RDF.

- The current GeoJSON draft in the IETF removes the possibility to specify and use any other CRS than CRS:84[2]. In our criteria the way CRS was proposed in geojson.org is too complicated. The issue could be resolved by adding an optional simple crs key in "geometry" that will contain a crs URI next to "coordinates".

- Only one geometry can be associated with a feature. Even if this limitation can be overcome by the use of a GeometricCollection, geometric properties have no name so there is no semantics associated to them. This could be also easily solved by adding a "propertyName" key to "geometry".

- Complex geometrical objects (other than points, lines and polygons) cannot be described.

- It does not support the raster/coverage model. This document proposes a solution for this in clause 9.

7.4 Proposing a WKT JSON for features

There was no consensus by the Testbed 11 participants in the convenience of defining an encoding that breaks compatibility with GeoJSON. GeoJSON has a high degree of acceptance in the geospatial community of developers and covers many use cases. Some people think that by defining a non-compatible JSON for feature encoding OGC risks dividing the community and reducing interoperability. Other people think that the limitations that current GeoJSON presents a need for an alternative solution.

After considering the limitation of the coordinates array discussed in the previous subclause, the proposed encoding proposes substituting the "coordinates" multidimensional array proposed in GeoJSON and use WKT (OGC 06-103r4) instead to describe the geometries of the features, allowing for an easy and automatic conversion of RDF to GeoJSON. The proposed encoding also includes support for different CRSs.

---

[2] This is an unfortunate result of discussions the axes order discussion between GeoJSON developers and some people that prefer following EPSG order. It is un unfortunate decision because,we have seen several examples of GeoJSON files using CRSs different from CRS84/WGS84 that could result invalid if IETF final approves GeoJSON the way it is.

The proposal replaces “geometry” in GeoJSON by a “literalGeometries” that can have a “free” name geometric complex properties that are formed by an id, a crs link and a wkt geometry as a text.

The example documented is a JSON-LD with a @context and feature collection. The @context section has two parts, one that will be generic for the format and another one that is common for a feature type. As a practical rule, I have avoided the use of the ‘@’ and ‘:” in the feature collection section (outside the @context section). By doing so, the access to objects and elements in JavaScript is simplified.

Generic part:

```
{
   "@context":
   {
      "xsd": "http://www.w3.org/2001/XMLSchema#",
      "geojson": "http://ld.geojson.org/vocab#",
      "schema": "http://schema.org/",
      "ogc_def": "http://www.opengis.net/def/",
      "ogc_geo": "http://www.opengis.net/ont/geosparql#",
      "wktjson": "http://www.opengis.net/wktjson/",

      "id": "@id",
      "url": "@id",
      "type": "@type",
      "features": "geojson:features",
      "properties": "_:",

      "featureType": "geojson:Feature",
      "featureCollectionType": "geojson:FeatureCollection",
      "lineStringType": "geojson:LineString",

      "Feature": "featureType",
      "FeatureCollection": "featureCollectionType",
      "LineString": "lineStringType",

      "literalGeometries":"_:",
      "asWKT": "ogc_geo:wktLiteral",
      "literalFeature": "wktjson:literalFeature",

      "crs": {
         "@id": "ogc_def:crs",
         "@type": "@id"
      },
```

One trick that we are using here is defining other characteristics of the feature types that are not going to be used and are going to be ignored by a common JSON-LD parser but could be useful to better describe the values of properties in the feature model. In this case we are providing the units of measure of the numeric properties.

```
      "uom": "ogc_def:uom",
```

Here we need to define a namespace for this feature type and an auxiliary namespace for the feature instances identifiers (in this case we are using the Wikipedia URL's as identifiers).

```
      "river": "http://www.opengis.uab.cat/River/",
      "wikipedia": "http://en.wikipedia.org/wiki/",
```

In GeoJSON there is no FeatureType concept but nothing prevents us to have a property with the name "@type" and with the value "riverType" for each feature instance.

```
      "riverType": "river",
```

These are the attributes of the instances of the "riverType" features. As you can see, it is possible to define a URL to a semantic definition, a data type and other properties that will be ignored by the JSON-LD parser.

```
      "name": "schema:name",
      "length": {
         "@id": "schema:distance",
         "@type": "xsd:float",
         "uom": "km"
      },
      "discharge": {
         "@id": "river:discharge",
         "@type": "xsd:float",
         "uom": "m^3/s"
      },
      "source": "river:source",
      "mouth": "river:mouth",
      "country": "schema:nationality",
      "bridges": "river:bridge",

      "riverExtremePos": "river:riverExtremePos"
   },
```

Once the @context part ends, the feature collection is presented. Each feature presents an id, a type, literal geometries and properties. In this example we present a feature collection formed by 2 features.

```
   "id": "river:ExampleRiverCollection",
   "type": "FeatureCollection",
   "features": [
   {
      "id": "wikipedia:Mississippi_River",
      "type": "literalFeature",
      "literalGeometries": {
         "id": "wikipedia:Mississippi_River",
```



```
         "riverExtremePos": {
            "id": "wikipedia:Mississippi_River/ExtremePos",
            "crs": "ogc_def:crs/OGC/1.3/CRS84",
            "asWKT": "LINESTRING ( -95.2075 47.239722, -89.253333
29.151111)"
         }
      },
      "properties": {
         "url": "wikipedia:Mississippi_River",
         "type": "riverType",
         "name": "mississipi",
         "length": 3734,
         "discharge": 16790,
         "source": "Lake Itasca",
         "mouth": "Gulf of Mexico",
         "country": "United States of America",
         "bridges": ["Eads Bridge", "Chain of Rocks Bridge"]
      }
   },
   {
      "id": "wikipedia:Ebro",
      "type": "literalFeature",
      "literalGeometries": {
         "url": "wikipedia:Ebro",
         "riverExtremePos": {
            "id": "wikipedia:Ebro/ExtremePos",
            "crs": "ogc_def:crs/OGC/1.3/CRS84",
            "asWKT": "LINESTRING (-4.402942 43.039111, 0.863056
40.72)"
         }
      },
      "properties": {
         "url": "wikipedia:Ebro",
         "type": "riverType",
         "name": "ebro",
         "length": 930,
         "discharge": 426,
         "source": "Pico Tres Mares",
         "mouth": "Mediterranean Sea",
         "country": "Spain"
      }
   }
   ]
}
```


The literalGeometries element replaces the “geometry” in a GeoJSON encoding and overcomes its main limitations. If we look in more detail at the literalGeometries element, we can see that it has the same id that the parent element (forcing the parser to

ignore the literalGeomtries grouping when translating it to RDF) and a list (one in this case) of geometric properties. These properties have an id, a crs uri and a literal wkt string.

```
      "literalGeometries": {
                "id": "wikipedia:Mississippi_River",
              "riverExtremePos": {
            "id": "wikipedia:Mississippi_River/ExtremePos",
            "crs": "ogc_def:crs/OGC/1.3/CRS84",
            "asWKT": "LINESTRING ( -95.2075 47.239722, -89.253333
29.151111)"
          }
      },
```

A JSON-LD parser is able to transform this document into RDF nquad notation.

```
<http://www.opengis.uab.cat/River/ExampleRiverCollection>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://ld.geojson.org/vocab#FeatureCollection> .
<http://www.opengis.uab.cat/River/ExampleRiverCollection>
      <http://ld.geojson.org/vocab#features>
      <http://en.wikipedia.org/wiki/Ebro> .
<http://www.opengis.uab.cat/River/ExampleRiverCollection>
      <http://ld.geojson.org/vocab#features>
      <http://en.wikipedia.org/wiki/Mississippi_River> .

<http://en.wikipedia.org/wiki/Ebro> <http://www.w3.org/1999/02/22-rdf-
      syntax-ns#type> <http://www.opengis.uab.cat/River/> .
<http://en.wikipedia.org/wiki/Ebro> <http://schema.org/distance>
      "930"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://en.wikipedia.org/wiki/Ebro> <http://schema.org/name> "ebro" .
<http://en.wikipedia.org/wiki/Ebro> <http://schema.org/nationality>
      "Spain" .
<http://en.wikipedia.org/wiki/Ebro>
      <http://www.opengis.uab.cat/River/discharge>
      "426"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://en.wikipedia.org/wiki/Ebro>
      <http://www.opengis.uab.cat/River/mouth> "Mediterranean Sea" .
<http://en.wikipedia.org/wiki/Ebro>
      <http://www.opengis.uab.cat/River/source> "Pico Tres Mares" .

<http://en.wikipedia.org/wiki/Ebro> <http://www.w3.org/1999/02/22-rdf-
      syntax-ns#type> <http://www.opengis.net/wktjson/literalFeature> .
<http://en.wikipedia.org/wiki/Ebro>
      <http://www.opengis.uab.cat/River/riverExtremePos>
      <http://en.wikipedia.org/wiki/Ebro/ExtremePos> .
<http://en.wikipedia.org/wiki/Ebro/ExtremePos>
      <http://www.opengis.net/def/crs>
      <http://www.opengis.net/def/crs/OGC/1.3/CRS84> .
<http://en.wikipedia.org/wiki/Ebro/ExtremePos>
      <http://www.opengis.net/ont/geosparql#wktLiteral> "LINESTRING (-
      4.402942 43.039111, 0.863056 40.72)" .
```



```
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.uab.cat/River/> .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://schema.org/distance>
      "3734"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://schema.org/name> "mississipi" .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://schema.org/nationality> "United States of America" .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.opengis.uab.cat/River/bridge> "Chain of Rocks Bridge"
      .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.opengis.uab.cat/River/bridge> "Eads Bridge" .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.opengis.uab.cat/River/discharge>
      "16790"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.opengis.uab.cat/River/mouth> "Gulf of Mexico" .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.opengis.uab.cat/River/source> "Lake Itasca" .

<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/wktjson/literalFeature> .
<http://en.wikipedia.org/wiki/Mississippi_River>
      <http://www.opengis.uab.cat/River/riverExtremePos>
      <http://en.wikipedia.org/wiki/Mississippi_River/ExtremePos> .
<http://en.wikipedia.org/wiki/Mississippi_River/ExtremePos>
      <http://www.opengis.net/ont/geosparql#wktLiteral> "LINESTRING ( -
      95.2075 47.239722, -89.253333 29.151111)" .
<http://en.wikipedia.org/wiki/Mississippi_River/ExtremePos>
      <http://www.opengis.net/def/crs>
      <http://www.opengis.net/def/crs/OGC/1.3/CRS84> .
```

7.5 GeoJSON-LD with a modified JSON-LD parser

Another possible solution is to introduce some modifications in the JSON-LD parser to transform GeoJSON coordinates into WKT on the fly resulting in a satisfactory RDF encoding. In fact, this solution was suggested in “https://github.com/geojson/geojson-ld/issues/31: “Develop and promote JSON-LD processor support for jsonld:jsonData”. This parser can be based in a WKT/GeoJSON JavaScript library such as http://openlayers.org/dev/examples/vector-formats.html, http://arthur-e.github.io/Wicket/, https://github.com/mapbox/wellknown or can be developed ad-hoc.

In this case, we use directly a GeoJSON file with the right @context document but we add some convenient elements to it that will be needed.

```
{
   "@context":
   {
```

```
      "xsd": "http://www.w3.org/2001/XMLSchema#",
      "geojson": "http://ld.geojson.org/vocab#",
      "schema": "http://schema.org/",
      "ogc_geo": "http://www.opengis.net/ont/geosparql#",

      "id": "@id",
      "url": "@id",
      "type": "@type",
      "features": "geojson:features",
      "geometry": "_:",
      "properties": "_:",

      "asWKT": "ogc_geo:wktLiteral",

      "featureType": "geojson:Feature",
      "featureCollectionType": "geojson:FeatureCollection",
      "lineStringType": "geojson:LineString",

      "Feature": "featureType",
      "FeatureCollection": "featureCollectionType",
      "LineString": "lineStringType",
      ...
   },
```

To demonstrate the feasibility of this approach, we have created a modified version of the http://json-ld.org/playground/index.html that includes a small piece of code that transforms the GeoJSON n-dimensional arrays of coordinates and the bbox into the WKT equivalents. The converted string is included as the value of the “asWKT” JSON key and the “coordinates” key is removed. The transformation is requested before the JSON-LD engine reads and interprets the JSON-LD file. The code to transform the coordinates array into WKT the following:

```
function geometryasWKT(geometry)
{
var identified=false;

   if (geometry.type=="Point")
   {
      geometry.asWKT="POINT(";
      for (var i_dim=0; i_dim<geometry.coordinates.length;
i_dim++)
      {
         geometry.asWKT+=geometry.coordinates[i_dim];
         if (i_dim+1<geometry.coordinates.length)
         geometry.asWKT+=" "
      }
      identified=true;
   }
```



```
   else if (geometry.type=="MultiPoint")
   {
      geometry.asWKT="MULTIPOINT(";
      for (var i_vrt=0; i_vrt<geometry.coordinates.length;
i_vrt++)
      {
         geometry.asWKT+="(";
         for (var i_dim=0;
i_dim<geometry.coordinates[i_vrt].length; i_dim++)
         {
            geometry.asWKT+=geometry.coordinates[i_vrt][i_dim];
            if (i_dim+1<geometry.coordinates[i_vrt].length)
            geometry.asWKT+=" "
         }
         geometry.asWKT+=")";
         if (i_vrt+1<geometry.coordinates.length)
            geometry.asWKT+=","
      }
      identified=true;
   }
   else if (geometry.type=="LineString")
   {
      geometry.asWKT="LINESTRING(";
      for (var i_vrt=0; i_vrt<geometry.coordinates.length;
i_vrt++)
      {
         for (var i_dim=0;
i_dim<geometry.coordinates[i_vrt].length; i_dim++)
         {
            geometry.asWKT+=geometry.coordinates[i_vrt][i_dim];
            if (i_dim+1<geometry.coordinates[i_vrt].length)
               geometry.asWKT+=" "
         }
         if (i_vrt+1<geometry.coordinates.length)
            geometry.asWKT+=","
      }
      identified=true;
   }
   else if (geometry.type=="MultiLineString" ||
geometry.type=="Polygon")
   {
      if (geometry.type=="MultiLineString")
         geometry.asWKT="MULTILINESTRING(";
      else
         geometry.asWKT="POLYGON(";
      for (var i_ls=0; i_ls<geometry.coordinates.length; i_ls++)
      {
         geometry.asWKT+="(";
         for (var i_vrt=0;
i_vrt<geometry.coordinates[i_ls].length; i_vrt++)
         {
```

```
            for (var i_dim=0;
i_dim<geometry.coordinates[i_ls][i_vrt].length; i_dim++)
            {

   geometry.asWKT+=geometry.coordinates[i_ls][i_vrt][i_dim];
               if
(i_dim+1<geometry.coordinates[i_ls][i_vrt].length)
                  geometry.asWKT+=" "
            }
            if (i_vrt+1<geometry.coordinates[i_ls].length)
               geometry.asWKT+=","
         }
         geometry.asWKT+=")";
         if (i_ls+1<geometry.coordinates.length)
            geometry.asWKT+=","
      }
      identified=true;
   }
   else if (geometry.type=="MultiPolygon")
   {
      geometry.asWKT="MULTIPOLYGON(";
      for (var i_pol=0; i_pol<geometry.coordinates.length;
i_pol++)
      {
          geometry.asWKT+="(";
          for (var i_ls=0;
i_ls<geometry.coordinates[i_pol].length; i_ls++)
          {
         geometry.asWKT+="(";
         for (var i_vrt=0;
i_vrt<geometry.coordinates[i_pol][i_ls].length; i_vrt++)
         {
            for (var i_dim=0;
i_dim<geometry.coordinates[i_pol][i_ls][i_vrt].length; i_dim++)
            {

   geometry.asWKT+=geometry.coordinates[i_pol][i_ls][i_vrt][i_dim
];
               if
(i_dim+1<geometry.coordinates[i_pol][i_ls][i_vrt].length)
                  geometry.asWKT+=" "
            }
            if (i_vrt+1<geometry.coordinates[i_pol][i_ls].length)
               geometry.asWKT+=","
         }
         geometry.asWKT+=")";
         if (i_ls+1<geometry.coordinates[i_pol].length)
            geometry.asWKT+=","
                  }
                  geometry.asWKT+=")";
```



```
                    if (i_pol+1<geometry.coordinates.length)
            geometry.asWKT+=","
         }
         identified=true;
      }
      //else if (geometry.type=="GeometryCollection")
      //    alert("GeometryCollection");
      if (identified==true)
      {
         geometry.asWKT+=")";
         delete geometry.coordinates;
         delete geometry.type;
      }
}

function bboxasWKT(feature)
{
   var n_dim=feature.bbox.length/2;

   feature.asWKT="POLYGON(((";
   for (var i_dim=0; i_dim<n_dim; i_dim++)
   {
      feature.asWKT+=feature.bbox[i_dim];
      if (i_dim+1<n_dim)
         feature.asWKT+=" "
   }
   feature.asWKT+="),(";
   for (var i_dim=0; i_dim<n_dim; i_dim++)
   {
      if (i_dim==1)
         feature.asWKT+=feature.bbox[n_dim+i_dim];
      else
         feature.asWKT+=feature.bbox[i_dim];
      if (i_dim+1<n_dim)
         feature.asWKT+=" "
   }
   feature.asWKT+="),(";
   for (var i_dim=0; i_dim<n_dim; i_dim++)
   {
      feature.asWKT+=feature.bbox[n_dim+i_dim];
      if (i_dim+1<n_dim)
         feature.asWKT+=" "
   }
   feature.asWKT+="),(";
   for (var i_dim=0; i_dim<n_dim; i_dim++)
   {
      if (i_dim==1)
         feature.asWKT+=feature.bbox[i_dim];
      else
         feature.asWKT+=feature.bbox[n_dim+i_dim];
      if (i_dim+1<n_dim)
```

```
         feature.asWKT+=" "
   }
   feature.asWKT+="),(";
   for (var i_dim=0; i_dim<n_dim; i_dim++)
   {
      feature.asWKT+=feature.bbox[i_dim];
      if (i_dim+1<n_dim)
         feature.asWKT+=" "
   }
   feature.asWKT+=")))";
   delete feature.bbox;
}

function geoJSONasWKT(geojson)
{
   if (typeof geojson == "object")
   {
     if (geojson.length)
     {
            for(var i=0; i<geojson.length; i++)
            {
         geoJSONasWKT(geojson[i]);
            }
     }
     else
     {
      for(var name in geojson)
      {
   if (geojson.hasOwnProperty(name))
   {
            if (name == "geometry" && typeof geojson[name] ==
"object")
            {
                if (geojson[name].type &&
geojson[name].coordinates)
                {
                    geometryasWKT(geojson[name]);
                }
            }
       else if (name == "geometries" && typeof geojson[name] ==
"object" && geojson[name].length)
       {
               for(var i=0; i<geojson[name].length; i++)
             {
                  if (geojson[name][i].type &&
geojson[name][i].coordinates)
                  {
               geometryasWKT(geojson[name][i]);
         }
                }
```



```
        }
        else if (name == "bbox" && typeof geojson[name] ==
"object" && geojson[name].length>3 && typeof geojson[name][0] ==
"number" && typeof geojson[name][1] == "number" && typeof
geojson[name][2] == "number" && typeof geojson[name][3] ==
"number")
        {
       bboxasWKT(geojson);
             }
             else if (typeof geojson[name] == "object")
       geoJSONasWKT(geojson[name]);
         }
       }
      }
    }
    return;
}
```

Bbox was converted to a rectangular polygon due to the fact that no equivalent element is present in WKT.

The result of the conversion is the same than in previous section and can be seen in the following illustration as a RDF graph:

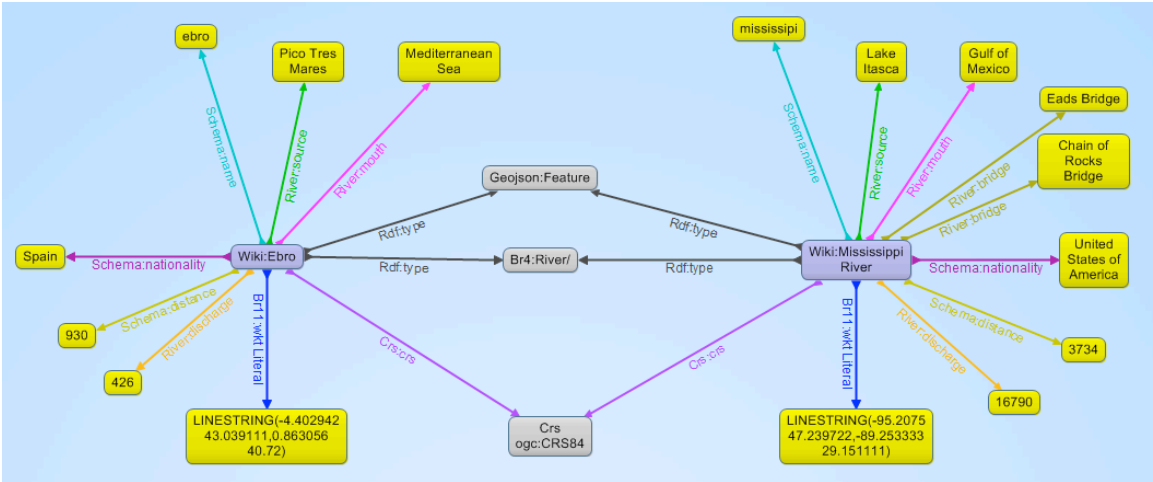

Figure 1: River GeoJSON file example transformed to RDF through JSON-LD

Recommendation 14: Respect the original format of GeoJSON and apply a piece of code to transform GeoJSON into WKT JSON for simple features on the fly to obtain RDF notation from GeoJSON.
Target: OAB

Recommendation 15: Consider JSON-LD as an alternative for creating GML application schemas as a means of defining feature types and as a mean for validation.
Target: OWS Common.SWG and OAB

Recommendation 16: Add a BBOX element in the WKT standard.
Target: CR to Simple Features for SQL

8 GeoJSON in OGC

8.1 OWS Context encoded in GeoJSON

The creation of a JSON encoding for OWS Context was started in the Testbed 10 and the results are collected in the Engineering Report: "OGC 14-009r1, OGC Testbed-10 Rules for JSON and GeoJSON Adoption: Focus on OWS-Context". A version for this standard candidate will soon be released for public comment with the reference OGC 14-055 OWS Context GeoJSON Encoding. What we did in the Testbed 11 was to create a JSON schema and a JSON-LD for validating the examples OWS Context group will provide with this standard.

8.1.1 JSON schema for OWS context JSON

Recommendation 17: Distribute this JSON schema, and the examples validated with it, in schemas.opengis.net when OWS context JSON standard gets approved.
Target: OWS Context

8.1.2 JSON-LD schema for OWS context JSON

Recommendation 18: Distribute this JSON-LD @context, and the examples validated with it, in schemas.opengis.net when OWS context JSON standard gets approved.
Target: OWS Common.SWG and OAB

8.2 Offer GeoJSON files in the web with OWS Context.

8.2.1 Simple approach based on "files"

Many people are starting to publish maps in GeoJSON by simply uploading files on GitHub. The Rioja Spatial Data Infrastructure in Spain is an example of this https://github.com/iderioja/base_datos_geografica/blob/master/README_EN.md giving access to official datasets in GeoJSON.

GitHub offers the users the capability to show the map directly in their portal and the versioning capability. Nevertheless, there is a general feeling that there is still a need for an easy discovery mechanism for geospatial data. Even if GitHub (and any other repository will not be able to do "subsetting" of the features, some common queries and subsets can also be stored. GitHub is also providing two other characteristics that make datasets more discoverable: Datasets are exposed by a list of automatic links and there is a manifest or metadata file that describes the data. This characteristic helps web search engines to better index the geospatial features: The data becomes discoverable.

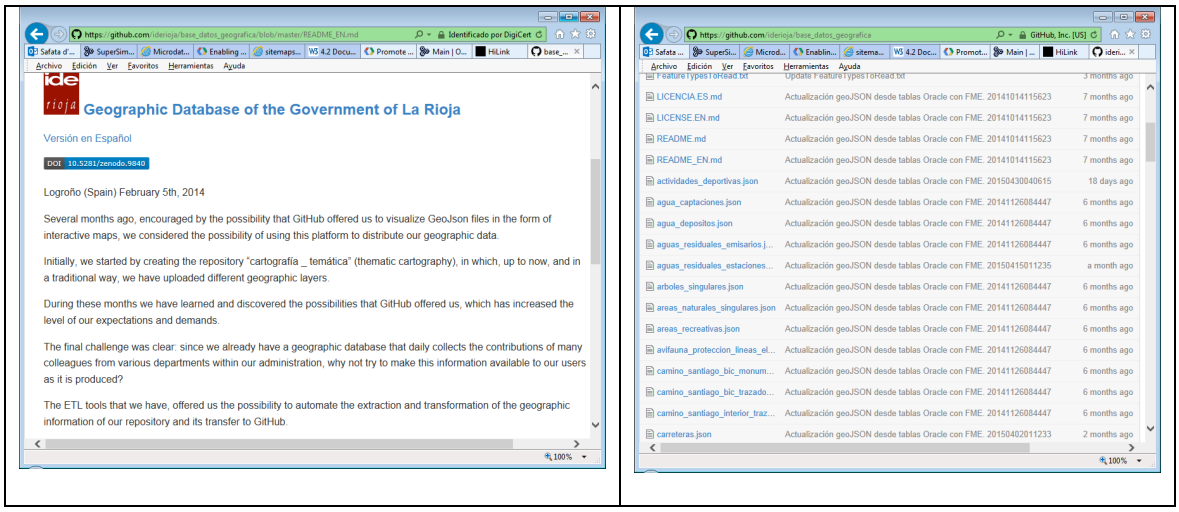

Figure 2: IDE Rioja using GitHub to provide geospatial information in GeoJSON

Could we make a standard mechanism of providing metadata about a set of links discoverable in the web? In fact, this is one of the original use cases considered in OWS Context. Indeed, OWS Context can provide a minimum set of metadata, and offer a link to a set of resources. In addition, each OWS Context resource can also link to a geospatial metadata description (e.g. a full ISO19115 description), to a schema, and to a web service to get the data in other formats or to add geospatial filtering capability. In other words, OWS context can be used as the single entry point to a list of geospatial resources.

In principle, we could use OWS context to expose and describe a list of GeoJSON files. Unfortunately, OWS context does not define any extension for GeoJSON linking or embedding. Even we could mimic other extensions to do the same it could be useful that OWS Context provides a specific extension for GeoJSON.

Recommendation 19: Create an extension of OWS Context JSON for illustrating how to reference GeoJSON data both embedded or linked.
Target: CR to OWS Context.SWG

8.2.2 Current mechanims in OWS Context.

Currently, OWS Context is only available in OWS Atom encoding. Very soon, the OWS Context SWG will release a GeoJSON encoding. What is the best way to expose OWS Context in the web? Atom-XML seemed a good alternative a few years ago but today, there are reasons to be concerned. There are some evidences that the main search engine in the web is removing RSS support. Recently, Google removed support to GeoRSS/Atom in Google maps. The new version of Google maps does not support Atom anymore even though it is still possible to request using the old version. How long Google will maintain support to the old version is uncertain. Google also removed an app called Google Reader that was a web based generic RSS reader. It is difficult to predict if this is also going to impact the Google search engine but there are reason to suspect that RSS is no longer of interest for Google.

The GeoJSON encoding seems a good alternative because we will have a homogeneous encoding where a GeoJSON-context will link to other GeoJSON files that will contain real (simple) features. The problem is that JSON doesn't provide native file linking mechanism and GeoJSON-context was forced to propose one (based on the transposition of the Atom link mechanism). The adoption of this or other linking mechanism on the web it not expected soon so it is not expected that this approach is useful in web search engines.

As we have demonstrated before, JSON-LD provides an alternative way of linking resources on the semantic web. Is it possible that search engines adopt some form of JSON or JSON-LD in the future? It seems so. In fact, Google is already experimenting with JSON-LD for describing critical reviews of web resources: https://developers.google.com/structured-data/critic-reviews. JSON-LD could be a promising alternative in the future.

8.2.3 HTML as a natural way for linking. OWS Context encoded in HTML

Everybody knows that HTML was designed with the linking capacity in mind. Both, users reading HTML and automatic crawlers, transverse links all the time. HTML seems the natural selection for linking JSON on the web. The question is how to complement the linking mechanism with some additional metadata that search engines could use for indexing. A solution could come from mechanism that web search engines already had agreed to use for better indexing: Microdata.

Microdata is a WHATWG[3] HTML specification that provides both vocabulary and strategies to embed metadata within existing content on web pages. Search engines, web crawlers, and browsers can extract and process Microdata from a web page and use it to provide a richer browsing experience for users (Google, Microsoft and Yahoo! rely on this markup). In particular, provide more relevant results to users. Microdata can also be considered an annotating mechanism in HTML elements with machine-readable tags. Microdata vocabularies provide the semantics, or meaning of an Item. The aim is to use Microdata to achieve better results in searches and also better presentations in the found items

Figure 3: A found recipe presentation in Google search results

[3] *Web Hypertext Application Technology Working Group*

A collection of commonly used markup vocabularies are provided by Schema.org schemas which include: Person, Event, Organization, Product, Review, Review-aggregate, Breadcrumb, Offer, Offer-aggregate. Where possible, authors are encouraged to re-use existing vocabularies, as this makes content re-use easier. It is also possible to create a custom vocabulary that better fits the purpose.

In this section we are going to explore the current use of common vocabularies to describe geospatial resources. Our purpose is to illustrate how this approach could work in practice and present the approach as a possible encoding for OWS Context in HTML. This section presents a solution that validates correctly in the Google Structured Data Testing Tool (https://developers.google.com/structured-data/testing-tool/). The possibility of defining a new vocabulary for describing geospatial resources can be considered later by the OWS Context group.

The simple use case that we are going to consider here is a list of resources that could be GeoJSON or TIFF files available on the web. To expose them we are going to create a web page that is going to link them. The common object in the Microdata vocabulary that better fits with OWS Context is the “Product” vocabulary. It is possible to create a web page that enumerates a list of products (http://schema.org/Product) and for each product it is possible to include one or more offers (http://schema.org/Offer). This way we are going to map the *product* concept to the *resource* concept in OWS Context and the *offer* concept to the *offering* concept in OWS Context.

One of the simplest examples in the currently approved OWS Context atom encoding is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<?valbuddy_schematron ../owc.sch?>
<?xml-model href="../atom.rnc" type="application/relax-ng-
regular-syntax"?>
<feed>
   <link rel="profile"
      href="http://www.opengis.net/spec/owc-atom/1.0/req/core"
      title="This file is compliant with version 1.0 of OGC
Context"/>
   <id>http://www.opengis.net/owc/1.0/examples/geotiff</id>
   <title>GeoTIFF Example</title>
   <subtitle type="html">
              GeoTIFF Example
   </subtitle>
   <author>
      <name>Joan Masó</name>
   </author>
   <updated>2012-11-04T17:26:23Z</updated>
   <entry>

   <id>ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg/ce
a.txt</id>
      <title>GeoTIFF Example</title>
```



```
      <updated>2011-11-01T00:00:00Z</updated>
      <dc:publisher>CREAF</dc:publisher>
      <content type="text">GeoTIFF Example coming from
ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg</content>
      <owc:offering code="http://www.opengis.net/spec/owc-
atom/1.0/req/geotiff">
         <owc:content type="image/tiff"
href="ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg/cea
.tif"/>
      </owc:offering>
   </entry>
</feed>
```

As any other OWS Context document it has two sections: the general metadata describing the context document and the list of resources. To encode the first section we can use the w3c standard for metadata (http://www.w3.org/TR/html5/document-metadata.html) and to encode the second part we will use Microdata.

The result is de following web page:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
  <link rel="profile"
      href="http://www.opengis.net/spec/owc-atom/1.0/req/core"
      title="This file is compliant with version 1.0 of OGC
Context"/>
  <title>GeoTIFF Example</title>
  <meta name="descriptor" content="A OWS Context GeoTIFF
Example">
  <meta name="keywords" content="geotiff,example">
  <meta name="author" content="Joan Masó">
  <meta name="atom:updated" content="2012-11-04T17:26:23Z">
</head>

<h1>GeoTIFF Example</h1>

<div itemscope itemtype="http://schema.org/Product">

  <h2><span itemprop="name">GeoTIFF Example 1</span></h2>
  <table border="0"><tr><td>
  <img itemprop="image" width="64" src="http://png-
3.findicons.com/files/icons/1637/file_icons_vs_2/256/tiff.png"
alt="GeoTIFF preview" /><br/></td><td>
  <span itemprop="brand">CREAF</span><br/>
  <span itemprop="description">GeoTIFF Example coming from
ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg</span><br
/>
```

```
  Product ID: <span
itemprop="productID">ftp://ftp.remotesensing.org/pub/geotiff/samp
les/gdal_eg/cea.txt</span><br/>
  </td></tr>
  </table>
  <h3>Offerings</h3>
  <span itemprop="offers" itemscope
itemtype="http://schema.org/Offer">
    <span itemprop="availableAtOrFrom" itemscope
itemtype="http://schema.org/Place">
          <div itemprop="geo" itemscope
itemtype="http://schema.org/GeoCoordinates">
      Latitude: 40 deg 44 min 54.36 sec N, Longitude: 73 deg 59
min 8.5 dec W
      <meta itemprop="latitude" content="40.75" />
      <meta itemprop="longitude" content="73.98" />
       </div>
    </span>

    Updated <time itemprop="availabilityStarts" datetime="2020-
11-05">2020-11-05</time><br/>
    Available from: <span itemprop="seller" itemscope
itemtype="http://schema.org/Organization">
                      <span itemprop="name">CREAF</span>
                    </span><br/>
    URL:<a
href="ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg/cea
.tif"
itemprop="url">ftp://ftp.remotesensing.org/pub/geotiff/samples/gd
al_eg/cea.tif</a><br/>
    Fees: <span itemprop="price" content="0.00">0.00</span>
  </span>
</div>
</html>
```

Note that we are also using http://schema.org/Place to encode a citation of a place. In schema.org a Place can be described in different ways including a physical postal address but also a “geo” element. A “geo” element can be a GeoCoordinates or a GeoShape. A GeoCoordinates is an element that describes the position as a single point that includes a longitude, a latitude and an elevation (and all the properties coming from Thing). On the contrary, GeoShape (see http://schema.org/GeoShape) provides 4 additional geometries to describe a Place: box, circle, line and polygon all accompanied by a single elevation. The description of the geometries is done in text and the restriction of the text is very simple and it does not specify the number of dimensions (2 are assumed) and coordinates are space separated. Polygons do not allow for holes and lines does not allow for multi-linestrings. Even if there are limitations in Place, it provides enough possibilities for the needs of the OWS Context where in many times, the geometric description of a resource requires just a bounding box or a simple envelop.

It is not always possible to make a one to one mapping between the OWS context and Microdata but many mappings are easily implemented. The current mapping was done making a priority that the Google Structured Data Testing Tool provides no errors.

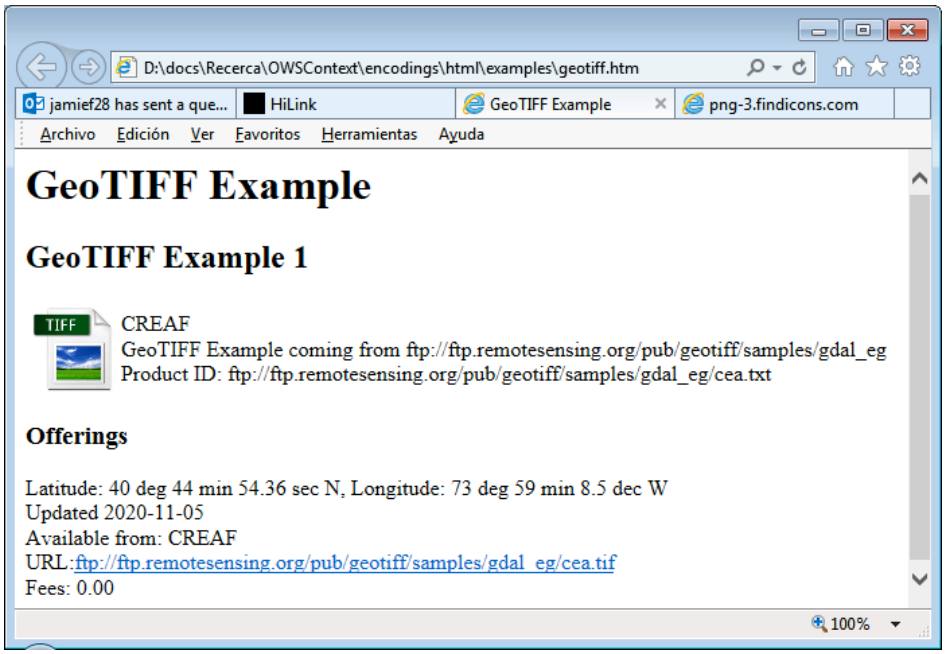

Figure 4: OWS Context example encoded in HTML

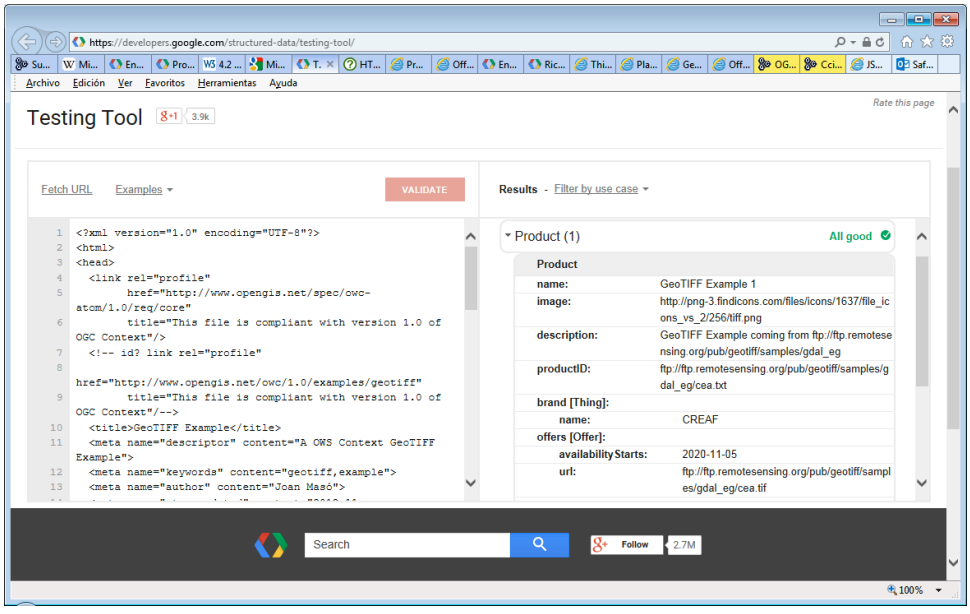

Figure 5: HTML OWS Context example tested in the Google Structured Data Testing Tool

Recommendation 20: Consider the HTML Microdata approach as a new encoding for OWS Context as a way to improve auto-discovery of OGC services and geospatial resources.
Target: OWS Context

Recommendation 21: Consider the benefits and drawbacks of extending schema.org vocabularies with a new type for "geospatial resource" that completely matches with OWS Context conceptual model.
Target: OWS Context

8.2.4 HTML as a natural way for linking. OWS Context encoded in JSON-LD in a webpage

You could wonder why the previous section has been included in this engineering report if it does not make use of JSON at all. There is a reason: schema.org vocabularies can be encoded in Microdata but also in JSON-LD. This way, the same information embedded in and <meta> tags of the previous example can also be encoded in a JSON-LD fragment that can be included in a web page.

This allows for proposing even another encoding for OWS Context based on schema.org vocabularies. We are still using the vocabularies for Products but we are encoding the same information in JSON-LD.

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Product",
  "description": "GeoTIFF Example coming from
ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg",
  "name": "GeoTIFF Example 1",
  "image": "http://png-
3.findicons.com/files/icons/1637/file_icons_vs_2/256/tiff.png",
  "brand": "CREAF",
  "productID":
"ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg/cea.txt"
,
  "offers": {
    "@type": "Offer",
    "availableAtOrFrom": {
   "@type": "Place",
   "geo": {
      "@type": "GeoCoordinates",
      "latitude": "40.75",
      "longitude": "73.98"
   }
    },
    "availabilityStarts": "2020-11-05",
    "seller": {
     "@type": "Organization",
     "name": "CREAF"
    },
```

```
    "url":
"ftp://ftp.remotesensing.org/pub/geotiff/samples/gdal_eg/cea.tif"
,
    "price": "0.00"
  }
}
</script>
```

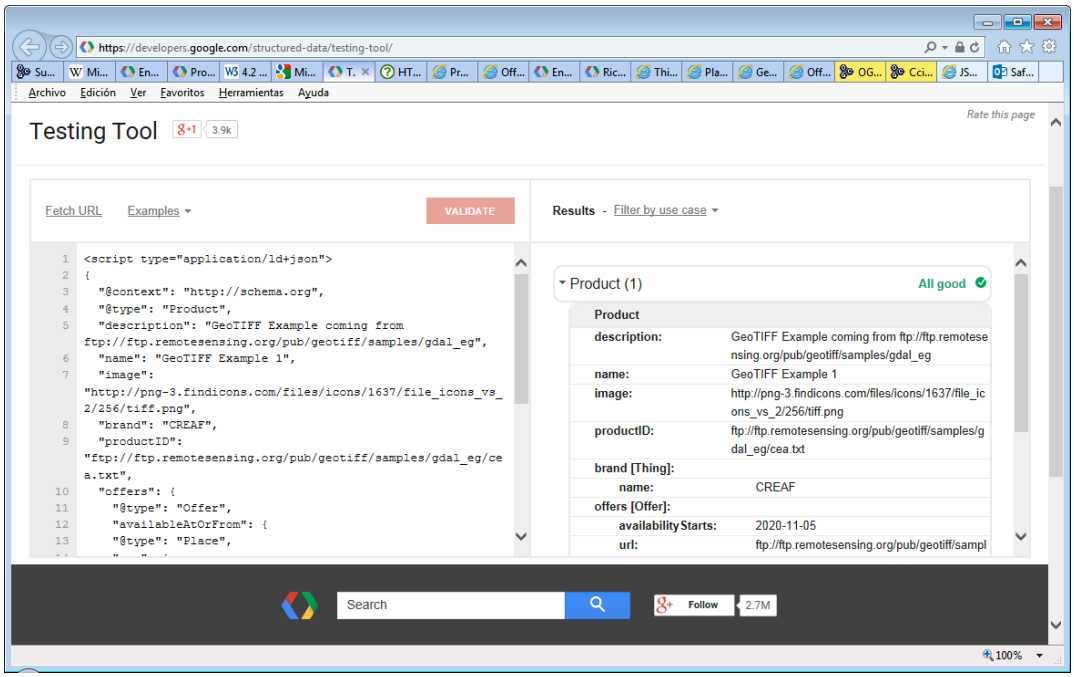

Figure 6: OWS Context example encoded in schema.org JSON-LD and tested in the Google Structured Data Testing Tool

Recommendation 22: Consider JSON-LD encodings for HTML structured data to create another encoding for OWS Context.
Target: OWS Context

9 Coverage JSON

GeoJSON covers the need for encoding features in JSON and makes it easier to deal with geospatial features in the browser. TopoJSON was created to easily encode and deal with specific case of 2D topological polygons in the browser. We could think if there is the same need for coverages. If we think just about referenced grids, a grid is just a sequence of values. In practice this could just be encoded in a JSON array. In fact, there is an initiative in Github to convert a netCDF file into JSON that just does a conversion of netCDF to JSON as an object that contains arrays of values: https://github.com/jllodra/ncdump-json

But a referenced grid is a bit more than an array. It requires some extra metadata to fully describe the meaning of the array of values. NetCDF also contains this description and the mentioned application is able also to extract it using this syntax:

ncdump-json sresa1b_ncar_ccsm3-example.nc -h –j

And the resulting data for a NetCDF example is:

```
{
   "dimensions":
   {
      "lat": 128,
      "lon": 256,
      "bnds": 2,
      "plev": 17,
      "time": "UNLIMITED ; // (1 currently)"
   },
   "variables":
   {
      "area":
      {
         "type": "float",
         "dimensions":
         [
            "lat",
            "lon"
         ],
         "attributes":
         {
            "long_name": "Surface area",
            "units": "meter2"
         }
      },
      "lat":
      {
         "type": "float",
         "dimensions":
         [
            "lat"
         ],
         "attributes":
         {
            "long_name": "latitude",
            "units": "degrees_north",
            "axis": "Y",
            "standard_name": "latitude",
            "bounds": "lat_bnds"
         }
      },
...
```

```
    },
    "global_attributes":
    {
        "CVS_Id": "$Id$",
        "creation_date": "",
        "prg_ID": "Source file unknown Version unknown Date
unknown",
        "cmd_ln": "bds -x 256 -y 128 -m 23 -o
/data/zender/data/dst_T85.nc",
        "contact": "ccsm@ucar.edu",
        "project_id": "IPCC Fourth Assessment",
        "Conventions": "CF-1.0",
        "references": "Collins, W.D., et al., 2005: The Community
Climate System Model, Version 3 Journal of Climate  Main website:
http://www.ccsm.ucar.edu",
        "acknowledgment": " Any use of CCSM data should acknowledge
the contribution of the CCSM project and CCSM sponsor agencies
...",
        "realization": 1,
        "experiment_id": "720 ppm stabilization experiment
(SRESA1B)",
        "model_name_english": "NCAR CCSM"
    }
}
```

This coverage description is specific for netCDF. There is a need for a standard description of a coverage in JSON.

9.1 GMLCov in JSON

OGC already provides a mean for encoding the description of a coverage that is raster encoding neutral and it is called GMLCov (OGC 09-146r2). This standard defines a coverage by providing metadata and data about the domainSet (definition of the grid structure), the rangeType (definition of the values meaning) and the rangeSet (the actual values).

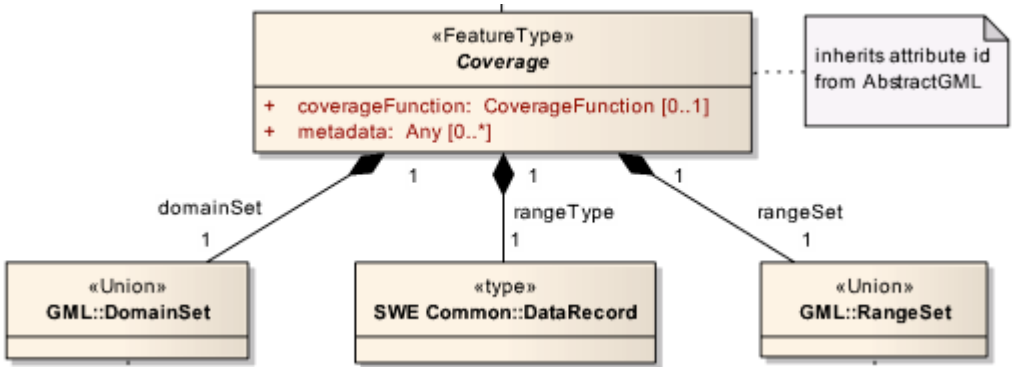

Figure 7: GMLCov main subclasses

Here we are applying the XML to JSON general rules to convert the GMLCov into a JSON file (with some exceptions). We are also proposing an example based on a dataset called etopo20 that presents the elevation and bathymetry: http://www.monsoondata.org:9090/dods/topo/rose/etopo20.info

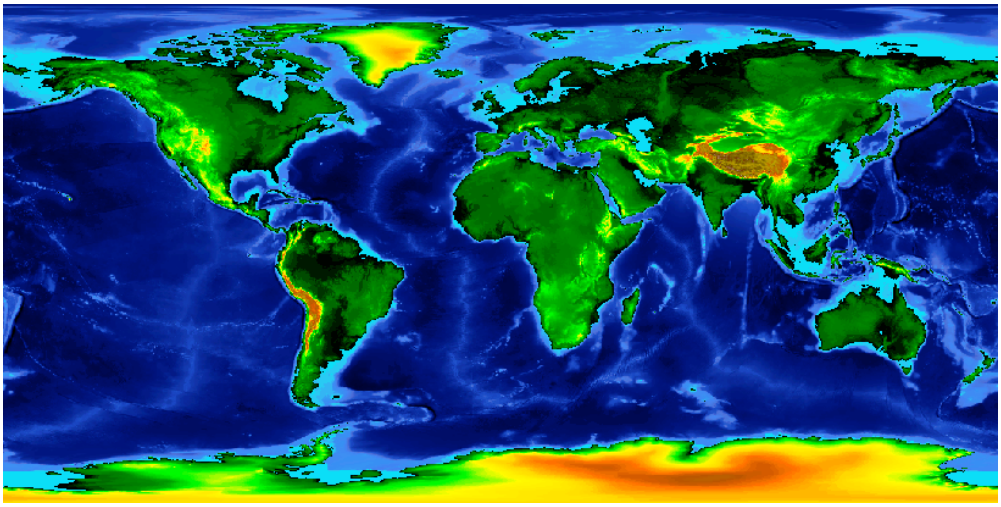

Figure 8: ETOPO20 dataset

The structure of the coverage JSON is:

```
{
    "type": "GridCoverage",
    "id": "http://www.someserver.com/examples/C0001",
    "bbox": [-180, -90, 180, 90],
    "domainSet": {...}
    "rangeSet": {...}
    "rangeType": {...}
}
```

The domainSet allow us to define the axis and their units, the pixel size, the origin and the number of rows and columns of the image. Please note the use of GeoJSON in some part.

```
    "domainSet":
    {
        "type:": "RectifiedGrid",
        "id": "http://www.someserver.com/examples/rg0001_C0001",
        "dimension": 2,
        "axisLabels": ["Long", "Lat"],
        "origin":
        {
            "geometry":
            {
                "crs":
                    "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
```

```
            "type": "Point",
            "coordinates": [ -180, -90]
         }
      },
      "offsetVectors": [ {
         "geometry":{
            "crs":
               "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
            "type": "Point",
            "coordinates": [ 0.0, 0.333333333333334]
         },
         "properties":{
            "axisLabel": "Long",
            "low": 0,
            "high": 1080
         }},{
         "geometry":{
            "crs":
"http://www.opengis.net/def/crs/OGC/1.3/CRS84",
            "type": "Point",
            "coordinates": [ -0.333333333333334, 0.0]
         },
         "properties":{
            "axisLabel": "Lat",
            "low": 0,
            "high": 540
         }
      }]
   },
```

The rangeType is useful for defining the semantics of the values in the coverage, its units and, also, the expected range of values.

```
   "rangeType":
   {

      "fields": [{

         "name": "elevation",
         "type": "quantity",
         "definition": "https://schema.org/elevation",
         "description": "Height",
         "uom":
         {
            "id": "http://www.opengis.net/def/uom/OGC/1.0/metre",
            "code": "m",
            "title": "meters"
         },
         "allowedValues":
         {
```

```
            "low": -7810,
            "high": 7000,
            "significantFigures": 4
         }
      }]
   }
}
```

For the rangeSet one immediate possibility could be to include the values directly as an array. We all know this is not a very efficient encoding so we could include a scale and translate values to scale and translate the values to a numeric range that is more compact in the same way that is suggested in topoJSON for the coordinates array.

```
   "rangeSet":

   {
      "dataBlocks":[
      {
         "name": "elevation",
         "scale": 1,
         "translate": 0,
         "values": [-4115,-4114,-4113,-4113,...,2783,2782,2782]
      }]
   },
```

We could wonder if this encoding is useful. The answer lies on the new characteristics of HTML5. The canvas new object allows for editing the individual pixel values of the canvas. This way it is possible to inject the coverage JSON values array into the canvas object and represent the data as an image without any client intervention. The following code fragment shows the data in a coverage JSON called “image” in a gray scale into a “map” canvas.

```
function DrawCoverage(map, image)
{
var i_cell=0, i_data=0, j, i, value256, a;

   var c=document.getElementById(map);
   if (c==null)
   {
      alert("No support for canvas detected");
      return;
   }
   c.width  = image.domainSet.offsetVectors[0].properties.high-
            image.domainSet.offsetVectors[0].properties.low;
   c.height = image.domainSet.offsetVectors[1].properties.high-
            image.domainSet.offsetVectors[1].properties.low;

   var ctx=c.getContext("2d");
   ctx.clearRect( 0, 0, ctx.canvas.width, ctx.canvas.height);
```

```
   var imgData=ctx.createImageData(c.width,c.height);

   a=256/(image.rangeType.fields[0].allowedValues.high -
         image.rangeType.fields[0].allowedValues.low);

   for (j=0;j<imgData.height;j++)
   {
      for (i=0;i<imgData.width;i++)
      {
         value256=Math.floor(a*(
            image.rangeSet.dataBlocks[0].values[i_cell] -
            image.rangeType.fields[0].allowedValues.low));
         imgData.data[i_data+0]=value256;
         imgData.data[i_data+1]=value256;
         imgData.data[i_data+2]=value256;
         imgData.data[i_data+3]=256;
         i_cell++;
         i_data+=4;
      }
   }
   ctx.putImageData(imgData,0,0);
}
```

As we already said, encoding raster values as text is not very efficient. Binary encodings are better. We can modify the rangeSet fragment to include a link to a binary file.

```
   "rangeSet":
   {
      "files":[
      {
         "name": "elevation",
         "scale": 1,
         "translate": 0,
         "fileName": "http://www.server.com/binary/etopo20.img"
      }]
   },
```

We could wonder if this encoding can be of any use in an Internet browser. Again there is an answer the new characteristics of HTML5. HTML5 has extended JavaScript capability with binary arrays. This means that it is now possible to download a binary file using AJAX, load it into a buffer array and then interpret it. The following function is able to download a binary file and deliver it:

```
function loadCoverageFile(path, success, error)
{
    var xhr = new XMLHttpRequest();
```



```
    xhr.onreadystatechange = function()
    {
        if (xhr.readyState === XMLHttpRequest.DONE) {
            if (xhr.status === 200) {
                if (success)
                    success(xhr.response);
            } else {
                if (error)
          error(xhr.statusText);
            }
        }
    };
    xhr.open("GET", path, true);
    xhr.responseType = "arraybuffer";

    xhr.send();
}
```

The following function loads it in the JavaScript representation of the coverage JSON:

```
function FileToDrawCoverage(ab)
{
   if (image.rangeSet.dataBlocks==null)
      image.rangeSet.dataBlocks=new Array(1);
   if (image.rangeSet.dataBlocks[0]==null)
      image.rangeSet.dataBlocks[0]=new Object();
   image.rangeSet.dataBlocks[0].arrayBuffer=ab;
}
```

The following function uses its values and sends them to the canvas. In this code we assume that the file has only int16 values (singed short int) for the values of the cells with no header and no compression. We use little endian byte order.

```
function DrawCoverage(map, image, palette, filter)
{
var i_cell=0, i_data=0, j, i, value, value256, a;

   var c=document.getElementById(map);
   if (c==null)
   {
      alert("No support for canvas detected");
      return;
   }
   c.width  = image.domainSet.offsetVectors[0].properties.high-
            image.domainSet.offsetVectors[0].properties.low;
   c.height = image.domainSet.offsetVectors[1].properties.high-
            image.domainSet.offsetVectors[1].properties.low;

   var ctx=c.getContext("2d");
   ctx.clearRect( 0, 0, ctx.canvas.width, ctx.canvas.height);
```

```
    var imgData=ctx.createImageData(c.width,c.height);
    var dv=new DataView(image.rangeSet.dataBlocks[0].arrayBuffer);

    a=256/(image.rangeType.fields[0].allowedValues.high -
          image.rangeType.fields[0].allowedValues.low);

    for (j=0;j<imgData.height;j++)
    {
       for (i=0;i<imgData.width;i++)
       {
          value=dv.getInt16(i_cell*2, littleEndian);
          value256=Math.floor(a*(value-
                image.rangeType.fields[0].allowedValues.low));
          imgData.data[i_data+0]=value256;
          imgData.data[i_data+1]=value256;
          imgData.data[i_data+2]=value256;
          imgData.data[i_data+3]=256;
          i_cell++;
          i_data+=4;
       }
    }
    ctx.putImageData(imgData,0,0);
}
```

A small demonstration based on the previous code is available at http://www.creaf.uab.cat/joanma/coveragejson/).

Coverage JSON in combination with HTML5 opens the possibility that a WCS serving coverage JSON can be used by a web browser client to show fragments of coverage data without the need of an intermediate image file or a WMS interface. The client will be able to change visualization options of even to make operations between coverages directly in the web browser because it has the actual data instead of a simple image.

Recommendation 23: Elaborate a converge JSON as a new standard encoding for GMLCov. Target: WCS.SWG

10 JSON in Web Services

10.1 JSON in OWS Common

For this section we are taking 06-121r9 OGC Web Services Common v.2.0.0 as a reference.

10.1.1 Service Metadata document in JSON

Subclause 7.4 in the OWS Common standard definesthe information that a GetCapabilities response (a service metadata document) has to contain. Fortunately, the model is defined in UML so it can be reused to elaborate an encoding for JSON. One of the interesting things in the transposition between the UML model and the XML is that UML model presents the property-object alternation (lowerCamelCase for properties and upperCamelCase for class names) but the translation into XML remove class names and always uses upperCamelCase. We suspect this was done to reduce the size and complexity of the encoded file instances. We believe that since JSON is less verbose we can reintroduce properties in lowerCamelCase and annotate class names as type keys as suggested in subclause Encoding the Object-property alternation in JSON.

Here we will review the service metadata document parts and will suggest encodings for each part based on the recommendations we have produced.

Recommendation 24: Include in OWS Common recommendations on how to provide service metadata in JSON derived from the UML models. Include as part of the OWS Common Schemas @context documents for independent validation of the 4 main sections of the Service Metadata. A JSON schema document can also be provided.
Target: OWS Common

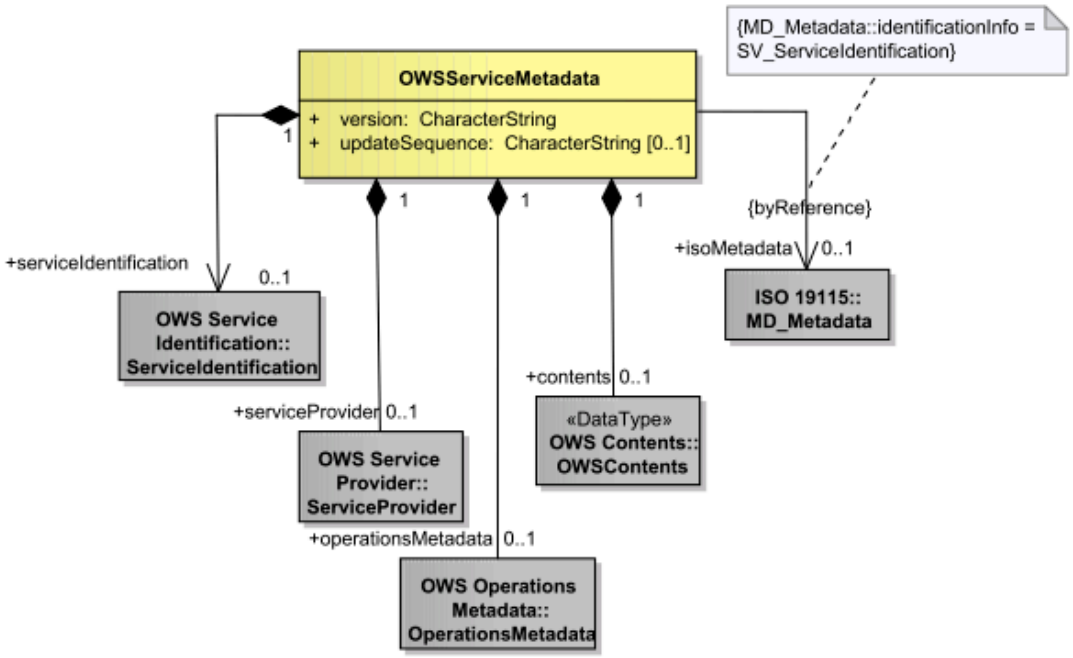

Figure 9: Service Metadata response UML diagram

The general structure is presented in JSON-LD as a @context and an example on how it looks like:

```
{
   "@context":
   {
      "ows": "http://www.opengis.net/ows/2.0/",
      "mywfs": "http://www.BlueOx.org/mywfs/2.5/",

      "id": "@id",
      "type": "@type",

      "serviceIdentification": "ows:serviceIdentification",
      "serviceProvider": "ows:serviceProvider",
      "operationsMetadata": "ows:operationsMetadata"
   },

   "id": "mywfs:Demo1",
   "type": "ows:ServiceMetadata",
   "serviceIdentification":
   {
      "id": "mywfs:ServiceIdentificationDemo1",
      "type": "ows:ServiceIdentification"
   },
   "serviceProvider":
   {
      "id": "mywfs:serviceProviderDemo1",
      "type": "ows:ServiceProvider"
   },
   "operationsMetadata":
   {
      "id": "mywfs:operationsMetadataDemo1",
      "type": "ows:OperationsMetadata"
   }
}
```

This can be automatically translated into an RDF representation:

```
<http://www.BlueOx.org/mywfs/2.5/Demo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/ServiceMetadata> .

<http://www.BlueOx.org/mywfs/2.5/Demo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification>
      <http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1> .
<http://www.BlueOx.org/mywfs/2.5/Demo1>
      <http://www.opengis.net/ows/2.0/serviceProvider>
      <http://www.BlueOx.org/mywfs/2.5/serviceProviderDemo1> .
<http://www.BlueOx.org/mywfs/2.5/Demo1>
      <http://www.opengis.net/ows/2.0/operationsMetadata>
      <http://www.BlueOx.org/mywfs/2.5/operationsMetadataDemo1> .
```



```
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/ServiceIdentification> .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/ServiceProvider> .
<http://www.BlueOx.org/mywfs/2.5/operationsMetadataDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/OperationsMetadata> .
```

10.1.1.1 ServiceIdentification in JSON

This is the service identification section presented in UML.

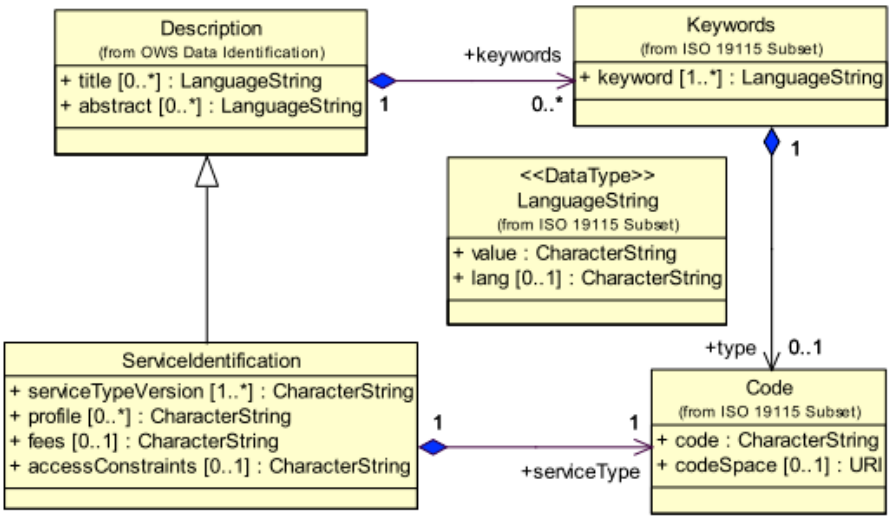

Figure 10: OWS Common ServiceIdentification UML diagram

The structure is presented in JSON-LD as a @context and an example:

```
{
   "@context":
   {
      "ows": "http://www.opengis.net/ows/2.0/",
      "mywfs": "http://www.BlueOx.org/mywfs/2.5/",

      "id": "@id",
      "type": "@type",

      "title": "ows:title",
      "abstract": "ows:abstract",
      "keywords": "_:",
      "keyword": "ows:keyword",

      "serviceType": "ows:serviceIdentification/serviceType",
```

```
      "serviceTypeVersion":
"ows:serviceIdentification/serviceTypeVersion",
      "fees": "ows:serviceIdentification/fees",
      "accessConstraints":
"ows:serviceIdentification/accessConstraints"
   },
   "id": "mywfs:ServiceIdentificationDemo1",
   "type": "ows:ServiceIdentification",

   "title": "OGC Member WFS",
   "abstract": "Web Feature Service maintained by NSDI data
provider, serving FGDC framework layer XXX; contact
Paul.Bunyon@BlueOx.org",
   "keywords":
   {
      "id": "mywfs:serviceIdentificationKeywordsDemo1",
      "type": "ows:Keywords",
      "keyword": ["FGDC", "NSDI", "Framework Data Layer"],
   },
   "serviceType": "WFS",
   "serviceTypeVersion": ["2.5.0", "2.0.0", "1.1.0", "1.0.0"],
   "fees": "NONE",
   "accessConstraints": "NONE"
}
```

That can be automatically transformed to RDF like this:

```
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/ServiceIdentification> .

<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/title> "OGC Member WFS" .
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/abstract> "Web Feature Service
      maintained by NSDI data provider, serving FGDC framework layer
      XXX; contact Paul.Bunyon@BlueOx.org" .

<http://www.BlueOx.org/mywfs/2.5/serviceIdentificationKeywordsDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/Keywords> .
<http://www.BlueOx.org/mywfs/2.5/serviceIdentificationKeywordsDemo1>
      <http://www.opengis.net/ows/2.0/keyword> "FGDC" .
<http://www.BlueOx.org/mywfs/2.5/serviceIdentificationKeywordsDemo1>
      <http://www.opengis.net/ows/2.0/keyword> "Framework Data Layer" .
<http://www.BlueOx.org/mywfs/2.5/serviceIdentificationKeywordsDemo1>
      <http://www.opengis.net/ows/2.0/keyword> "NSDI" .

<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification/serviceType>
      "WFS" .
```



```
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification/serviceTypeV
      ersion> "1.0.0" .
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification/serviceTypeV
      ersion> "1.1.0" .
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification/serviceTypeV
      ersion> "2.0.0" .
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification/serviceTypeV
      ersion> "2.5.0" .
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification/accessConstr
      aints> "NONE" .
<http://www.BlueOx.org/mywfs/2.5/ServiceIdentificationDemo1>
      <http://www.opengis.net/ows/2.0/serviceIdentification/fees> "NONE"
      .
```

10.1.1.2 ServiceProvider in JSON

This is the service provider section presented in UML.

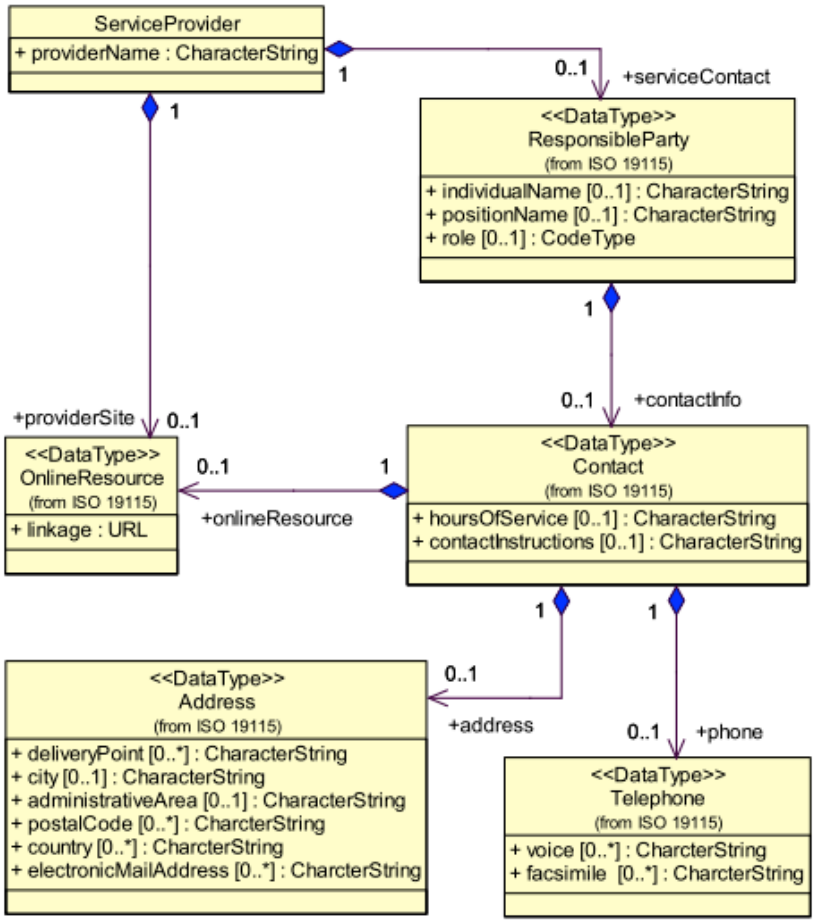

Figure 11: OWS Common ServiceProvider UML diagram

The structure is presented in JSON-LD as a @context and an example:

```
{
   "@context":
   {
      "ows": "http://www.opengis.net/ows/2.0/",
      "mywfs": "http://www.BlueOx.org/mywfs/2.5/",

      "id": "@id",
      "type": "@type",

      "providerName": "ows:serviceProvider/providerName",
      "providerSite":
      {
         "@id": "ows:serviceProvider/providerSide",
         "@type": "@id"
      },
      "serviceContact": "ows:serviceProvider/serviceContact:",
      "individualName": "ows:serviceProvider/individualName",
```

```
      "positionName": "ows:serviceProvider/positionName",
      "contactInfo": "ows:serviceProvider/contactInfo:",
      "phone": "ows:serviceProvider/phone:",
      "voice": "ows:serviceProvider/phone-Voice",
      "facsimile": "ows:serviceProvider/phone-Facsimile",
      "address": "ows:serviceProvider/addresss:",
      "deliveryPoint": "ows:serviceProvider/deliveryPoint",
      "city": "ows:serviceProvider/City",
      "administrativeArea":
"ows:serviceProvider/administrativeArea",
      "postalCode": "ows:serviceProvider/postalCode",
      "country": "ows:serviceProvider/country",
      "electronicMailAddress":
"ows:serviceProvider/electronicMailAddress",
      "onlineResource":
      {
         "@id": "ows:serviceProvider/onlineResource",
         "@type": "@id"
      },
      "hoursOfService": "ows:serviceProvider/hoursOfService",
      "contactInstructions":
"ows:serviceProvider/contactInstructions",
      "role": "ows:serviceProvider/role"
   },
   "id": "mywfs:serviceProviderDemo1",
   "type": "ows:ServiceProvider",
   "providerName": "BlueOx Inc.",
   "providerSite": "http://www.cubewerx.com",
   "serviceContact":
   {
      "id": "mywfs:serviceProviderServiceContactDemo1",
      "type": "ows:ServiceContact",
      "individualName": "Paul Bunyon",
      "positionName": "Mythology Manager",
      "contactInfo":
      {
         "id": "mywfs:serviceProviderContactInfoDemo1",
         "type": "ows:Contact",
         "phone":
         {
            "id": "mywfs:serviceProviderPhoneDemo1",
            "type": "ows:Telephone",
            "voice": "1.800.BIG.WOOD",
            "facsimile": "1.800.FAX.WOOD"
         },
         "address":
         {
            "id": "mywfs:serviceProviderAddressDemo1",
            "type": "ows:Address",
            "deliveryPoint": "North Country",
            "city": "Small Town",
```

```
            "administrativeArea": "Rural County",
            "postalCode": "12345",
            "country": "USA",
            "electronicMailAddress": "Paul.Bunyon@BlueOx.org"
         },
         "onlineResource": "http://www.BlueOx.org/contactUs",
         "hoursOfService": "24x7",
         "contactInstructions": "eMail Paul with normal requests;
Phone Paul for emergency requests; if you get voice mail and your
request can't wait, contact another mythological figure listed on
the contactUs page of our web site."
      },
      "role": "PointOfContact"
   }
}
```

That can be automatically transformed to RDF like this:

```
<http://www.BlueOx.org/mywfs/2.5/serviceProviderDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/ServiceProvider> .

<http://www.BlueOx.org/mywfs/2.5/serviceProviderDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/providerName>
      "BlueOx Inc." .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/providerSide>
      <http://www.cubewerx.com> .

<http://www.BlueOx.org/mywfs/2.5/serviceProviderServiceContactDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/individualName>
      "Paul Bunyon" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderServiceContactDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/positionName>
      "Mythology Manager" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderServiceContactDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/role>
      "PointOfContact" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/serviceContact:>
      <http://www.BlueOx.org/mywfs/2.5/serviceProviderServiceContactDemo
      1> .

<http://www.BlueOx.org/mywfs/2.5/serviceProviderContactInfoDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/Contact> .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderServiceContactDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/ServiceContact> .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderContactInfoDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/addresss:>
      <http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1> .
```



```
<http://www.BlueOx.org/mywfs/2.5/serviceProviderContactInfoDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/contactInstruction
      s> "eMail Paul with normal requests; Phone Paul for emergency
      requests; if you get voice mail and your request can't wait,
      contact another mythological figure listed on the contactUs page
      of our web site." .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderContactInfoDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/hoursOfService>
      "24x7" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderContactInfoDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/onlineResource>
      <http://www.BlueOx.org/contactUs> .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderContactInfoDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/phone:>
      <http://www.BlueOx.org/mywfs/2.5/serviceProviderPhoneDemo1> .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderServiceContactDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/contactInfo:>
      <http://www.BlueOx.org/mywfs/2.5/serviceProviderContactInfoDemo1>
      .

<http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/Address> .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/City> "Small Town"
      .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/administrativeArea
      > "Rural County" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/country> "USA" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/deliveryPoint>
      "North Country" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/electronicMailAddr
      ess> "Paul.Bunyon@BlueOx.org" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderAddressDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/postalCode>
      "12345" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderPhoneDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/phone-Facsimile>
      "1.800.FAX.WOOD" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderPhoneDemo1>
      <http://www.opengis.net/ows/2.0/serviceProvider/phone-Voice>
      "1.800.BIG.WOOD" .
<http://www.BlueOx.org/mywfs/2.5/serviceProviderPhoneDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/Telephone> .
```

10.1.1.3 OperationsMetadata in JSON

This is the operations metadata section presented in UML.

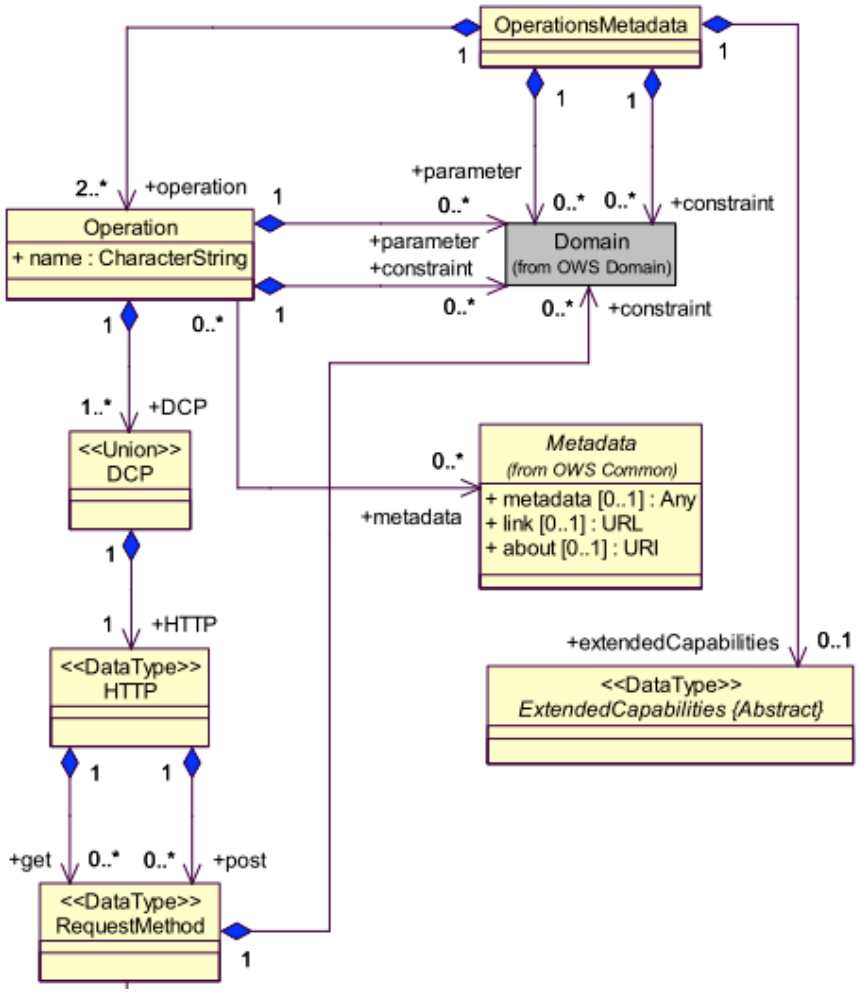

Figure 12: OWS Common OperationsMetadata UML diagram

The structure is presented in JSON-LD as a @context and an example:

```
{
   "@context":
   {
      "ows": "http://www.opengis.net/ows/2.0/",
      "mywfs": "http://www.BlueOx.org/mywfs/2.5/",

      "id": "@id",
      "name": "@id",
      "type": "@type",

      "operation": "ows:OperationsMetadata/Operation",
      "DCP": "_:",
      "HTTP": "_:",
      "Get": "_:",
      "href":
```

```
      {
         "@id": "ows:href",
         "@type": "@id"
      },
      "noValues": "ows:noValues",
      "defaultValue": "ows:defaultValue",
      "constraints": "ows:constraint"
   },
   "id": "mywfs:operationsMetadataDemo1",
   "type": "ows:OperationsMetadata",
   "operation": [
   {
      "name": "mywfs:operationDemo1/GetCapabilities",
      "type": "ows:Operation",
      "DCP": [
      {
         "HTTP":
         {
            "Get":
            {
               "name":
"mywfs:operationDemo1/GetCapabilities/method",
               "type": "ows:RequestMethod",
               "href": "http://www.BlueOx.org/mywfs/2.5?"
            }
         }
      }]
   },
   {
      "name": "mywfs:OperationDemo1/NoOp",
      "type": "ows:Operation",
      "DCP": [
      {
         "HTTP":
         {
            "Get":
            {
               "name": "mywfs:OperationDemo1/NoOp/method",
               "type": "ows:RequestMethod",
               "href": "http://www.BlueOx.org/mywfs/2.5?"
            }
         }
      }]
   } ],
   "constraints": [
   {
      "name":
"mywfs:OperationsMetadata/Constraint/ImplementsBasicWFS",
      "type": "ows:Domain",
      "noValues": "",
      "defaultValue": "TRUE"
```

```
   },          {
      "name":
"mywfs:OperationsMetadata/Constraint/ImplementsTransactionalWFS",
      "type": "ows:Domain",
      "noValues": "",
      "defaultValue": "TRUE"
   }]
}
```

That can be automatically transformed to RDF like this:

```
<http://www.BlueOx.org/mywfs/2.5/operationsMetadataDemo1>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/OperationsMetadata> .

<http://www.BlueOx.org/mywfs/2.5/operationsMetadataDemo1>
      <http://www.opengis.net/ows/2.0/OperationsMetadata/Operation>
      <http://www.BlueOx.org/mywfs/2.5/operationDemo1/GetCapabilities> .
<http://www.BlueOx.org/mywfs/2.5/operationsMetadataDemo1>
      <http://www.opengis.net/ows/2.0/OperationsMetadata/Operation>
      <http://www.BlueOx.org/mywfs/2.5/OperationDemo1/NoOp> .

<http://www.BlueOx.org/mywfs/2.5/OperationDemo1/NoOp>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/Operation> .
<http://www.BlueOx.org/mywfs/2.5/OperationDemo1/NoOp/method>
      <http://www.opengis.net/ows/2.0/href>
      <http://www.BlueOx.org/mywfs/2.5?> .
<http://www.BlueOx.org/mywfs/2.5/OperationDemo1/NoOp/method>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/RequestMethod> .

<http://www.BlueOx.org/mywfs/2.5/operationDemo1/GetCapabilities>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/Operation> .
<http://www.BlueOx.org/mywfs/2.5/operationDemo1/GetCapabilities/method>
      <http://www.opengis.net/ows/2.0/href>
      <http://www.BlueOx.org/mywfs/2.5?> .
<http://www.BlueOx.org/mywfs/2.5/operationDemo1/GetCapabilities/method>
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/RequestMethod> .

<http://www.BlueOx.org/mywfs/2.5/operationsMetadataDemo1>
      <http://www.opengis.net/ows/2.0/constraint>
      <http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Imp
      lementsBasicWFS> .
<http://www.BlueOx.org/mywfs/2.5/operationsMetadataDemo1>
      <http://www.opengis.net/ows/2.0/constraint>
      <http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Imp
      lementsTransactionalWFS> .

<http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Implemen
      tsBasicWFS> <http://www.opengis.net/ows/2.0/defaultValue> "TRUE" .
<http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Implemen
      tsBasicWFS> <http://www.opengis.net/ows/2.0/noValues> "" .
```



```
<http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Implemen
      tsBasicWFS> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.opengis.net/ows/2.0/Domain> .

<http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Implemen
      tsTransactionalWFS> <http://www.opengis.net/ows/2.0/defaultValue>
      "TRUE" .
<http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Implemen
      tsTransactionalWFS> <http://www.opengis.net/ows/2.0/noValues> "" .
<http://www.BlueOx.org/mywfs/2.5/OperationsMetadata/Constraint/Implemen
      tsTransactionalWFS> <http://www.w3.org/1999/02/22-rdf-syntax-
      ns#type> <http://www.opengis.net/ows/2.0/Domain> .
```

10.1.1.4 Contents section in Service Metadata in JSON

The Contents section of the service metadata document is more ambiguous in OWS Common because it depends deeply on the nature of the service. OWS Common only specifies that a list of resource descriptions will be includes (if calls each one "datasetSummary"). A dataset summary should have an id, a title, an abstract, some keywords, a Bounding Box in WGS84 and Bounding Boxes in other CRS's. All this elements are also present in OWS Context. This ER is proposing to use a OWS Context encodings for this section. Since OWS Context in GeoJSON has already been drafted, we thing that adopting it will help to reduce redundancies and increase interoperability.

Recommendation 25: Consider OWS Context as the *Contents* section of service metadata document. In particular adopt the OWS Context JSON encoding in the JSON encoding of OWS Common.
Target: OWS Common

10.1.2 JSON GetCapabilities request

Subclause 7.2.4 on OWS Common is considering a XML encoding for GetCapabilities. In our opinion, we should avoid creating a JSON request for GetCapabilities and limit JSON usage in combination with REST requests (or KVP request).

Recommendation 26: OWS Common should recommend a REST requests for GetCapabilities and should recommend JSON as a default request.
Target: OWS Common

10.1.3 JSON requests

For complex requests, we may consider the need for a JSON encoding. In this case, OWS Common defines a minimum set of parameters in subclause 9.2.1. OWS Common should provide a @context fragment defining them.

Recommendation 27: Define the OWS Common minimum set of parameters in a request as a @context fragment. A JSON schema can also be provided.
Target: OWS Common

10.1.4 JSON exception

Subclause 7.4.1 on OWS Common defines an exception report for GetCapabilities and Clause 8 the general rules of the exceptions. In our opinion, we should avoid creating a JSON exception for GetCapabilities and limit ourselves to use the HTTP error levels and appropriate descriptions for them in a more RESTful style.

10.1.5 JSON responses

JSON responses can be provided also when currently an XML response is possible. In this case, the standard will provide a @context fragment and eventually a JSON Schema.

10.1.6 Bounding Boxes

Subclause 10.2 on OWS Common defines how to use Bounding Boxes in OWS Common. SubClause 10.2.2 and in the case of JSON encoding whenever possible we should use GeoJSON bbox.

Subclause 10.2.1 defines a generic bounding box.

```
"CRSbbox":
{
   "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
   "LowerCorner": "POINT(-180 -90)",
   "UpperCorner": "POINT(180 90)"
}
```

10.2 JSON in Web Map Services

This subclause discusses how to include JSON in the WMS service.

10.2.1 JSON in GetCapabilities response

The subclause 10.1.1 already discuses this topic in a more general way.

10.2.2 JSON for WMS GetMap response

In the past, several discussions suggested the need to be able on how to get metadata about individual maps returned by a WMS service. A JSON response can be a opportunity to do so:

To request a JSON response in GetMap you only have to request the right format:

```
"?service=WMS&request=GetMap&format=application/json"
```

Then, the response can contain some information about the map and the map image itself embedded or linked.

The following example return the image code embedded in the JSON as base64 encoding

```
{
    "id": 15,
    "box": [-180,-90,180,90],
    "title": "Example data fragment",
    "author": "ACME Corp.",
    "img":{
        "format": "image/gif",
        "data": "iVBORw...kJggg=="
    }
}
```

The following example return the image code embedded in the JSON as base64 encoding

```
{
    "id": 16,
    "title": "Example data fragment",
    "author": "ACME Corp.",
    "img": {
        "href"="?service=WMS&request=GetMap&format=image/gif"
    }
}
```

This encoding ideas reproduce some suggestion coming from: http://json-schema.org/latest/json-schema-hypermedia.html.

Another possibility is return GeoJSON vectors and relies on the ability of the map brower to present the information to the client. For example Leaflet map client is able to render GeoJSON on the screen.

10.2.3 JSON for WMS GetFeatureInfo response

GetFeatureInfo response format was left open to the implementations. This was done to make it simple. Since in WMS the data model associated with the features that are represented in the map is not exposed (at least in the general case with no SLD support), we cannot rely on the feature type concept. In this case the capability of the JavaScript language and JSON of being able to allow for any set of properties can play in our advantage.

It is important that we support both FeatureInfo reports coming from any kind of features including grid coverages. In a general implementation, information can be provided not just about the features were the I,J point is contained but also from its surroundings. In fact some WMS implementations have a “distance” parameter to specify the radius of the circle where the server look inside for features to return information about them (see support for the vendor specific parameter “buffer” in geoserver 2.7 http://docs.geoserver.org/stable/en/user/services/wms/vendor.html).

We propose a special use of GeoJSON to respond to GetFeatureInfo. The GeoJSON will contain a collection of features where each one will represent a feature in the original data. In the “geometry” part, we return a GeoJSON point when the feature was covering or touching the I,J position or a GeoJSON string formed by two 2 vertices that goes from the CRS values of the point where the place I,J where pointing and a point that is in the interior or the border of the returned feature. In the “properties” part we could return the properties of this feature.

```
{
   "type": "FeatureCollection",
   "features":
   [
      {
         "type": "Feature",
         "geometry":
         {
            "type": "Point",
            "coordinates":
            [-47.314159, 45.3141519],
         },
         "id": 20245062,
         "properties":
         {
            "COMID": 20245062,
            "FDATE": "1999-11-24T06:00:00Z",
            "LENGTHKM": 5.415,
            "REACHCODE": "16060014044574"
         }
      }
   ]
}
```

Please note that we recommend that features are well identified. We are not recommending returning the full geometry description because we assume that this is the work of a WFS GetFeature operation using identifiers recovered by the GetFeatureInfo request.

Recommendation 28: In WMS 1.4 include a encoding for GetFeatureInfo responses based on GeoJSON but replacing the geometry part by maker of the position of the query and the position of the returned feature. If returned objects correspond to simple features, return an “id” that allows recovering the geometry using an additional WFS query.
Target: WMS.SWG

10.3 JSON in Web Map Tile Service

10.3.1 JSON in Map Tile Service

It is worth mentioning that a very simple open specification (linked to what it seems a single vendor) called tileJSON has already been produced to describe a set of tiles in JSON and can be found here: https://github.com/mapbox/tilejson-spec/tree/master/2.1.0.

In essence, it provides a very simple encoding for describing the availability of a WMTS Simple profile layer in Web Mercator.

The minimum tileJSON file is:

```
{
  "tiles": [ "http://tile.openstreetmap.org/{z}/{x}/{y}.png" ],
  "minzoom": 0,
  "maxzoom": 18
}
```

One of the good thing of this encoding is that this JSON fragment can be ingested directly in web browser code. This is a code that generates a map browser for the San Francisco area:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8 />
  <title></title>
  <script
src='https://api.tiles.mapbox.com/mapbox.js/v2.1.9/mapbox.js'></s
cript>
  <link
href='https://api.tiles.mapbox.com/mapbox.js/v2.1.9/mapbox.css'
rel='stylesheet' />
  <style>
    body { margin:0; padding:0; }
    .map { position:absolute; top:0; bottom:0; width:100%; }
  </style>
</head>
<body>
<div id='map' class='map'> </div>

<script>
var tilejson = {
  "tiles": [ "http://tile.openstreetmap.org/{z}/{x}/{y}.png" ],
  "minzoom": 0,
  "maxzoom": 18
}
L.mapbox.map('map', tilejson, {
    scrollWheelZoom: false
```

```
}).setView([37.82053680, -122.36481177], 11);
</script>

</body>
</html>
```

Recommendation 29: Consider to include tileJSON in a non normative example for the WMTS Simple profile.
Target: WMS.SWG

10.3.2 JSON encoding for a TileMatrixSet

One of the elements that makes singular the WMTS is capability to completely describe a tile matrix set. Being able to provide the description of a tile matrix set as in JSON could simplify the way WMTS clients deal with tile space descriptions. In particular it could be useful to have a json description of the fixed tile matrix sets provided by the new WMTS simple profile.

```
{
   "tileMatrixSet": [ {
      "id": "wmtss:WorldWebMercatorQuad",
      "type": "wmts:TileMatrixSet",
      "title": "Google Maps Compatible for the World",
      "CRSbbox":
      {
         "crs": "http://www.opengis.net/def/crs/EPSG/0/3857",
         "lowerCorner": "POINT(-20037508.3427892, -
20037508.3427892)",
         "upperCorner": "POINT(20037508.3427892,
20037508.3427892)"
      },
      "crs": "http://www.opengis.net/def/crs/EPSG/0/3857",
      "wellKnownScaleSet":
"http://www.opengis.net/def/wkss/OGC/1.0/GoogleMapsCompatible",

      "tileMatrix": [ {
         "id": "wmtss:WorldWebMercatorQuad0",
         "type": "wmts:TileMatrix",
         "scaleDenominator": 559082264.0287178,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 1,
         "matrixHeight": 1
      },{
         "id": "wmtss:WorldWebMercatorQuad1",
         "type": "wmts:TileMatrix",
         "scaleDenominator": 279541132.0143589,
```



```
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 2,
         "matrixHeight": 2
      },{
         "id": "wmtss:WorldWebMercatorQuad18",
         "type": "wmts:TileMatrix",
         "scaleDenominator": 2132.729583849784,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 262272,
         "matrixHeight": 262272
      } ]
   }]
}
```

Recommendation 30: Include in the WMTS Simple profile the JSON description of the WMTS simple two tile matrix sets.
Target: WMS.SWG

10.4 Serving GeoJSON with a Web Feature Service

Some people believe that WFS can only serve GML features but even if this is a common case nothing in the standard prevents from using other geospatial formats and encodings. In particular, it is possible to server GeoJSON as a result of a WFS GetFeature request. In fact, GeoServer has been doing this for a quite a while. A more profound question is if it possible to have a WFS without GML support. WFS is deeply related with the idea of the GML application schema and the definition of feature types. We already have mentioned that GeoJSON does not provide this concept directly. Reintroducing this concept in GeoJSON as an extension is easy by means of adding to each GeoJSON feature a property called “type”. This will allow a WFS server to group the features in feature types and to provide a list of available feature types in the GetCapabilities.

Recommendation 31: For a WFS serving GeoJSON, force the features to have a property that contains the feature type.
Target: WFS.SWG

A fundamental question could be the format that a DescribeFeatureType operation has to return when asking the characteristics of a feature type. Having to return a GML encoding seems unnecessary. One possibility explored is to return a encoding neutral representation for describing feature types as described in ISO19110. Following the new ISO19115-3 encoding for metadata a river object can be encoded like this:

```
<gfc:FC_FeatureCatalogue>
   <gfc:featureType>
      <gfc:FC_FeatureType>
         <gfc:typeName>riverType</gfc:typeName>
```

```
         <gfc:isAbstract>
            <gco:Boolean>false</gco:Boolean>
         </gfc:isAbstract>
         <gfc:carrierOfCharacteristics>
            <gfc:FC_FeatureAttribute>
               <gfc:memberName> length </gfc:memberName>
               <gfc:definition><gco:CharacterString>Length
</gco:CharacterString></gfc:definition>
               <gfc:cardinality>1</gfc:cardinality>
               <gfc:code><gco:CharacterString>length
</gco:CharacterString></gfc:code>
               <gfc:valueMeasurementUnit>m
               </gfc:valueMeasurementUnit>
            </gfc:FC_FeatureAttribute>
         </gfc:carrierOfCharacteristics>
      </gfc:FC_FeatureType>
   </gfc:featureType>
</gfc:FC_FeatureCatalogue>
```

It could be preferable to be able use a JSON encoding also for this aspect. There is always the possibility to generate a JSON encoding for ISO19110 but it could be better to adopt a solution that already exists. We have already discussed in this ER some of the current alternatives for GeoJSON validation in subclause 7 that could be used here. We have explored the possibility to create specific JSON schema for each feature type and also to use JSON-LD "@context" section to achieve a similar functionality. We have seen that JSON-LD is an approved standard and also allows a nice connection to RDF representations of features. The connection to RDF and the semantic web is particularly attractive giving the use of GeoJSON and extra characteristic that GML cannot provide as easily.

Recommendation 32: Consider that WFS DescribeFeatureType returns a @context section describing the Feature type in GeoJSON.
Target: WFS.SWG

10.4.1 Pointers to GeoJSON fragments

One of the issues that we have in using GeoJSON for describing features is that WFS uses FES, and FES needs XPath:

```
<fes:PropertyIsLessThanOrEqualTo>
   <fes:ValueReference>myns:Person/myns:mailAddress/
            myns:Address/myns:streetNumber</fes:ValueReference>
   <fes:Literal>10999</fes:Literal>
</fes:PropertyIsLessThanOrEqualTo>
```

Currently, JSON provides 2 alternatives to do this: JSON Pointer (https://tools.ietf.org/html/rfc6901) and JSON Path (http://goessner.net/articles/JsonPath).

JSON pointer is used to specify a specific node in a JSON encoded document while JSON Path mimics most of the XPath query functionalities.

In JSON pointer the last XML fragment will look like:

```
"@context": "http://www.opengis.net/fes/2.0",
"PropertyIsLessThanOrEqualTo": {
   "ValueReference": "/Person/mailAddress/Address/streetNumber",
   "Literal": 10999
}
```

In JSON Path the last XML fragment will look like:

```
"@context": "http://www.opengis.net/fes/2.0",
"PropertyIsLessThanOrEqualTo": {
   "ValueReference": "$.Person.mailAddress.Address.streetNumber",
   "Literal": 10999
}
```

Recommendation 33: Consider using either JSON Pointer or JSON Path in places where a XPath is required.
Target: WFS.SWG

10.5 Metadata in JSON

In this subclause we discuss aspects about the official metadata provided mainly in ISO 19115 data model by the producer of the geospatial resource and geospatial user feedback produced by the consumer of the resource.

10.5.1 ISO Metadata in JSON

This aspect has not been developed in this testbed but some effort has been found in the web. One of the most significants is:
https://github.com/adiwg/mdBook/blob/master/mdjson_schemas/README.md.

10.5.2 Geospatial User Feedback in JSON

During the review of the schema.org materials we realize that there is already an standard for user feedback in the Internet. Google is providing support to it to shop user reviews directly on the search results page:

Figure 13: Google search result showing a rating average.

The rating comes from the actual page internal structure (http://www.imdb.com/title/tt0382625/) encoded in Microdata that we show here in a simplified form:

```
<div itemtype="http://schema.org/AggregateRating" itemscope
itemprop="aggregateRating">
Ratings:
<span itemprop="ratingValue">6.5</span>/<span
itemprop="bestRating">10</span></span> from
<span itemprop="ratingCount">278,569</span> users
Reviews:
<span itemprop="reviewCount">1,943 user</span>|
<span itemprop="reviewCount">291 critic</span>|
</div>
```

Figure 14: How IMDb presents user feedback to users

Indeed, Microdata is used everywhere in this page to define the movie itself, the director, etc as can be seen in this html fragment.

```
<div id="pagecontent" itemscope
itemtype="http://schema.org/Movie">
<span class="itemprop" itemprop="name">The Da Vinci Code</span>
<time itemprop="duration" datetime="PT149M" >149 min</time>
<meta itemprop="datePublished" content="2006-05-19" />
<div class="txt-block" itemprop="director" itemscope
itemtype="http://schema.org/Person">...
</div></div>
```

The exact encoding of AggregateRating in Microdata and in JSON-LD can be seen at: http://schema.org/AggregateRating.

Table 4: Elements of the AgregateRating

Name[b]	Definition	Data type and value	Multiplicity and use
itemReviewed	The item that is being reviewed/rated.	Thing	Unspecified
ratingCount	The count of total number of ratings.	Integer	Unspecified
reviewCount	The count of total number of reviews.	Integer	Unspecified
bestRating[a]	The highest value allowed in this rating system. If bestRating is omitted, 5 is assumed.	Text or Number	Unspecified
ratingValue[a]	The rating for the content.	Text	Unspecified
worstRating[a]	The lowest value allowed in this rating system. If worstRating is omitted, 1 is assumed.	Text or Number	Unspecified

[a] Properties inherited from Rating

[b] Other properties are inherited from Thing, See http://schema.org/Thing

In many cases, feedback is accompanied AggregateRating and Ratings are acompanied by information of individual Reviews as specified in http://schema.org/Review.

Table 5: Elements of the Review

Name	Definition	Data type and value	Multiplicity and use
itemReviewed	The item that is being reviewed/rated.	Thing	Unspecified
reviewBody	The actual body of the review.	Text	Unspecified
reviewRating	The rating given in this review. Note that reviews can themselves be rated. The reviewRating applies to rating given by the review. The	Rating	Unspecified

Name	Definition	Data type and value	Multiplicity and use
	aggregateRating property applies to the review itself, as a creative work.		
about	The subject matter of the content.	Thing	Unspecified
accessibilityAPI	Indicates that the resource is compatible with the referenced accessibility API (WebSchemas wiki lists possible values).	Text	Unspecified
accessibilityControl	Identifies input methods that are sufficient to fully control the described resource (WebSchemas wiki lists possible values).	Text	Unspecified
accessibilityFeature	Content features of the resource, such as accessible media, alternatives and supported enhancements for accessibility (WebSchemas wiki lists possible values).	Text	Unspecified
accessibilityHazard	A characteristic of the described resource that is physiologically dangerous to some users. Related to WCAG 2.0 guideline 2.3 (WebSchemas wiki lists possible values).	Text	Unspecified
accountablePerson	Specifies the Person that is legally accountable for the CreativeWork.	Person	Unspecified
aggregateRating	The overall rating, based on a collection of reviews or ratings, of the item.	AggregateRating	Unspecified
alternativeHeadline	A secondary title of the CreativeWork.	Text	Unspecified
associatedMedia	A media object that encodes this CreativeWork. This property is a synonym for encoding.	MediaObject	Unspecified
audience	An intended audience, i.e. a group for whom something was created.	Audience	Unspecified

Name	Definition	Data type and value	Multiplicity and use
	Supersedes serviceAudience.		
audio	An embedded audio object.	AudioObject	Unspecified
author	The author of this content. Please note that author is special in that HTML 5 provides a special mechanism for indicating authorship via the rel tag. That is equivalent to this and may be used interchangeably.	Person or Organization	Unspecified
award	An award won by or for this item. Supersedes awards.	Text	Unspecified
character	Fictional person connected with a creative work.	Person	Unspecified
citation	A citation or reference to another creative work, such as another publication, web page, scholarly article, etc.	CreativeWork or Text	Unspecified
comment	Comments, typically from users.	Comment	Unspecified
commentCount	The number of comments this CreativeWork (e.g. Article, Question or Answer) has received. This is most applicable to works published in Web sites with commenting system; additional comments may exist elsewhere.	Integer	Unspecified
contentLocation	The location depicted or described in the content. For example, the location in a photograph or painting.	Place	Unspecified
contentRating	Official rating of a piece of content—for example,'MPAA PG-13'.	Text	Unspecified
contributor	A secondary contributor to the CreativeWork.	Person or Organization	Unspecified

Name	Definition	Data type and value	Multiplicity and use
copyrightHolder	The party holding the legal copyright to the CreativeWork.	Person or Organization	Unspecified
copyrightYear	The year during which the claimed copyright for the CreativeWork was first asserted.	Number	Unspecified
creator	The creator/author of this CreativeWork. This is the same as the Author property for CreativeWork.	Person or Organization	Unspecified
dateCreated	The date on which the CreativeWork was created.	Date	Unspecified
dateModified	The date on which the CreativeWork was most recently modified.	Date	Unspecified
datePublished	Date of first broadcast/publication.	Date	Unspecified
discussionUrl	A link to the page containing the comments of the CreativeWork.	URL	Unspecified
editor	Specifies the Person who edited the CreativeWork.	Person	Unspecified
educationalAlignment	An alignment to an established educational framework.	AlignmentObject	Unspecified
educationalUse	The purpose of a work in the context of education; for example, 'assignment', 'group work'.	Text	Unspecified
encoding	A media object that encodes this CreativeWork. This property is a synonym for associatedMedia. Supersedes encodings.	MediaObject	Unspecified
exampleOfWork	A creative work that this work is an example/instance/realization/derivation of. Inverse property: workExample.	CreativeWork	Unspecified
genre	Genre of the creative work or group.	Text	Unspecified

Name	Definition	Data type and value	Multiplicity and use
hasPart	Indicates a CreativeWork that is (in some sense) a part of this CreativeWork. Inverse property: isPartOf.	CreativeWork	Unspecified
headline	Headline of the article.	Text	Unspecified
inLanguage	The language of the content or performance or used in an action. Please use one of the language codes from the IETF BCP 47 standard. Supersedes language.	Language or Text	Unspecified
interactivityType	The predominant mode of learning supported by the learning resource. Acceptable values are 'active', 'expositive', or 'mixed'.	Text	Unspecified
isBasedOnUrl	A resource that was used in the creation of this resource. This term can be repeated for multiple sources. For example, http://example.com/great-multiplication-intro.html.	URL	Unspecified
isFamilyFriendly	Indicates whether this content is family friendly.	Boolean	Unspecified
isPartOf	Indicates a CreativeWork that this CreativeWork is (in some sense) part of. Inverse property: hasPart.	CreativeWork	Unspecified
keywords	Keywords or tags used to describe this content. Multiple entries in a keywords list are typically delimited by commas.	Text	Unspecified
learningResourceType	The predominant type or kind characterizing the learning resource. For example, 'presentation', 'handout'.	Text	Unspecified
license	A license document that applies to this content, typically indicated by URL.	CreativeWork or URL	Unspecified
mainEntity	Indicates the primary entity described in some page or other CreativeWork.	Thing	Unspecified

Name	Definition	Data type and value	Multiplicity and use
	Inverse property: mainEntityOfPage.		
mentions	Indicates that the CreativeWork contains a reference to, but is not necessarily about a concept.	Thing	Unspecified
offers	An offer to provide this item—for example, an offer to sell a product, rent the DVD of a movie, or give away tickets to an event.	Offer	Unspecified
position	The position of an item in a series or sequence of items.	Integer or Text	Unspecified
producer	The person or organization who produced the work (e.g. music album, movie, tv/radio series etc.).	Person or Organization	Unspecified
provider	The service provider, service operator, or service performer; the goods producer. Another party (a seller) may offer those services or goods on behalf of the provider. A provider may also serve as the seller. Supersedes carrier.	Person or Organization	Unspecified
publication	A publication event associated with the item.	Publication Event	Unspecified
publisher	The publisher of the creative work.	Organization	Unspecified
publishingPrinciples	Link to page describing the editorial principles of the organization primarily responsible for the creation of the CreativeWork.	URL	Unspecified
recordedAt	The Event where the CreativeWork was recorded. The CreativeWork may capture all or part of the event. Inverse property: recordedIn.	Event	Unspecified
releasedEvent	The place and time the release was issued, expressed as a	Publication Event	Unspecified

Name	Definition	Data type and value	Multiplicity and use
	PublicationEvent.		
review	A review of the item. Supersedes reviews.	Review	Unspecified
schemaVersion	Indicates (by URL or string) a particular version of a schema used in some CreativeWork. For example, a document could declare a schemaVersion using an URL such as http://schema.org/version/2.0/ if precise indication of schema version was required by some application.	URL or Text	Unspecified
sourceOrganization	The Organization on whose behalf the creator was working.	Organization	Unspecified
text	The textual content of this CreativeWork.	Text	Unspecified
thumbnailUrl	A thumbnail image relevant to the Thing.	URL	Unspecified
timeRequired	Approximate or typical time it takes to work with or through this learning resource for the typical intended target audience, e.g. 'P30M', 'P1H25M'.	Duration	Unspecified
translator	Organization or person who adapts a creative work to different languages, regional differences and technical requirements of a target market.	Person or Organization	Unspecified
typicalAgeRange	The typical expected age range, e.g. '7-9', '11-'.	Text	Unspecified
version	The version of the CreativeWork embodied by a specified resource.	Number	Unspecified
video	An embedded video object.	VideoObject	Unspecified
workExample	Example/instance/realization/derivation of the concept of this creative work. eg. The paperback edition, first	CreativeWork	Unspecified

Name	Definition	Data type and value	Multiplicity and use
	edition, or eBook. Inverse property: exampleOfWork.		
additionalType	An additional type for the item, typically used for adding more specific types from external vocabularies in microdata syntax. This is a relationship between something and a class that the thing is in. In RDFa syntax, it is better to use the native RDFa syntax - the 'typeof' attribute - for multiple types. Schema.org tools may have only weaker understanding of extra types, in particular those defined externally.	URL	Unspecified
alternateName	An alias for the item.	Text	Unspecified
description	A short description of the item.	Text	Unspecified
image	An image of the item. This can be a URL or a fully described ImageObject.	URL or ImageObject	Unspecified
mainEntityOfPage	Indicates a page (or other CreativeWork) for which this thing is the main entity being described. Inverse property: mainEntity.	CreativeWork or URL	Unspecified
name	The name of the item.	Text	Unspecified
potentialAction	Indicates a potential Action, which describes an idealized action in which this thing would play an 'object' role.	Action	Unspecified
sameAs	URL of a reference Web page that unambiguously indicates the item's identity. E.g. the URL of the item's Wikipedia page, Freebase page, or official website.	URL	Unspecified
url	URL of the item.	URL	Unspecified

[a] Properties inherited from Rating

Name	Definition	Data type and value	Multiplicity and use
[b] Other properties are inherited from Thing, See http://schema.org/Thing			

The page https://developers.google.com/structured-data/critic-reviews provides information on how Google has implemented it an some examples on how this is implements in JSON-LD, the encoding strongly recommended by Google.

```
<script type="application/ld+json">
{
  "@context":"http://schema.org",
  "@type":"Review",
  "author":{
    "@type":"Person",
    "name":"Lisa Kennedy",
    "sameAs":"https://plus.google.com/114108465800532712602"
  },
  "datePublished":"2014-03-13T20:00",
  "description":"Nerve-racking, sentimental and thrilling.",
  "itemReviewed":{
    "@type":"Movie",
    "name":"Gravity",
    "sameAs":"http://www.imdb.com/title/tt1454468/",
    "datePublished":"2013-10-04T00:00",
    "director":{
      "@type":"Person",
      "name":"Alfonso Cuarón",
      "sameAs":"http://en.wikipedia.org/wiki/Alfonso_Cuar%C3%B3n"
    },
    "actor":[
      {
        "@type":"Person",
        "name":"Sandra Bullock",
        "sameAs":"http://en.wikipedia.org/wiki/Sandra_Bullock"
      },
      {
        "@type":"Person",
        "name":"George Clooney",
        "sameAs":"http://en.wikipedia.org/wiki/George_Clooney"
      }
    ]
  },
  "publisher":{
    "@type":"Organization",
    "name":"Denver Post",
    "sameAs":"http://www.denverpost.com"
  },
  "reviewRating":{
    "@type":"Rating",
    "worstRating":1,
    "bestRating":4,
    "ratingValue":3.5
  },
  "url":"http://www.denverpost.com/movies/ci_24225964/gravity-
movie-review-anchored-by-sandra-bullock-its"
}
</script>
```


Recommendation 34: Consider the schema.org model for describing a geospatial user feedback model
Target: GUF.SWG

Recommendation 35: Consider a JSON-LD encoding for Geospatial User Feedback that maps the conceptual model to the http://schema.org/AggregateRating and http://schema.org/Review elements
Target: GUF.SWG

Recommendation 36: Consider including experimentation in GUF and OWS Context using microdata format in JSON-LD for geospatial features.
Target: Testbed-12

11 Rules for encoding JSON-LD from UML

During the presentation of a draft of this document at the 2015 Boulder OGC TC there was general agreement in the value of having a set of rules to derive a JSON-LD encoding from a UML model (as a better alternative than of deriving the JSON from the XML directly). This clause has the intention of summarizing the rules applied throughout this document. The aim is to provide the necessary material for discussion and to create a more formal OGC document later. Currently this set of rules is not fully comprehensive. The missing aspects are emphasized.

11.1 Property name limitations

11.2 Rules for simple data types

11.2.1 Text encoding

Unfortunately, there is no way to specify the character set used in a JSON encoded file. Worse than that, when JSON is included in an HTML file that has been produced and is marked in a character set, some implementation consider that the JSON file is in the same encoding that the HTML that embed it. Obviously, a JSON file can be included from more than one HTML than are marked with different character set, resulting in misinterpretations of the character set of the JSON file. To avoid that, https://tools.ietf.org/html/rfc7159, considers that "a JSON file SHALL be encoded in UTF-8, UTF-16, or UTF-32. The default encoding is UTF-8, and JSON texts that are encoded in UTF-8 are interoperable in the sense that they will be read successfully by the maximum number of implementations (there are many implementations that cannot successfully read texts in other encodings (such as UTF-16 and UTF-32))". "Implementations MUST NOT add a byte order mark to the beginning of a JSON text.",

11.2.2 Number encoding

https://tools.ietf.org/html/rfc7159 says that "A number is represented in base 10 using decimal digits. A number contains an integer component that may be prefixed with an optional minus sign, which may be followed by a fraction part and/or an exponent part. A fraction part is a decimal point followed by one or more digits. An exponent part begins with the letter 'E' in upper or lower case, which may be followed by a plus or minus sign.

The E and optional sign are followed by one or more digits. Infinity and NaN are not permitted (remember that “null” is allowed). There are not theoretical limitations on number figures BUT two limits can be assumed: for floating point numbers, many software follow IEEE 754-2008 binary64 (double precision) numbers. In addition, for integers numbers, it is not recommendable to go beyond the range [–(2**53)+1, (2**53)–1].

11.2.3 Simple data types in JSON-LD

JSON-LD syntax associates simple types to simple data types in the xsd namespace. It explicitly mentions xsd:string (that is the default value), xsd:double or as xsd:integer for numbers, and xsd:boolean (for the true or false values).

Even if it is not mentioned explicitly. Other types xsd can be used, and in particular, xsd:dateTime and xsd:duration are mentioned in implementations of schema.org (https://github.com/json-ld/json-ld.org/blob/master/contexts/person.jsonld) and are recommended here. This way, data types in UML can be easily mapped to this xsd types.

11.2.4 Identifiers, URLs and URI in JSON-LD

“@id” is a reserved type for URI’s and URL’s and all URI’s and URL’s should use this type.

11.2.5 Declaration of simple data types

In the “@context” section you should define the data types of all properties that are going to be used. To do that, we associate a key in the JSON file to a property name in the data model and to a simple data type like this:

```
"@context": {
   "xsd": "http://www.w3.org/2001/XMLSchema#",

   "onlineResource": {
      "@id": "ows:onlineResource", "@type": "@id"}
   "date": {
      "@id": "ows:publication", "@type": "xsd:dateTime"}
}
```

A list of equivalence between the UML simple types and the xsd time could be useful here. It will look very similar to the one we use in the UML XML conversions.

11.3 Rules for complex data types

Complex data types are defined with namespaces. Each namespace has a URI and an abbreviation. In JSON-LD, and abbreviated namespace is defined as a synonymous of a URI namespace in a @context section like this:


```
"@context": {
   "ows": "http://www.opengis.net/ows/2.0/",
}
```

Having a namespace for our own types and id's is also useful

```
"@context": {
   "ows": "http://www.opengis.net/ows/2.0/",
   "mythings": "http://www.someserver.org/mythings/1.0/",
}
```

Ending the namespace with a"/" is useful because the namespace string is ready to be concatenated with any property name, class name, or data type name.

We assume that all datatypes in this namespace are well defined in this namespace.

11.3.1 Listing you property names

In the "@context" section you should list all property names that are going to be used. To do that, we associate a key in the JSON file to a property name in the UML data model like this (note that no mention of the data types or classes is done here):

```
"@context": {
   "ows": "http://www.opengis.net/ows/2.0/",
   "serviceIdentification": "ows:serviceIdentification",
   "serviceProvider": "ows:serviceProvider",
   "operationsMetadata": "ows:operationsMetadata"
}
```

11.3.2 Declaring complex data types

The declaration of simple data types is done in a @context section. The declaration of a complex data type is done directly in the object instances using the reserved property name "@type". Types can contain the abbreviated namespace. In addition, each object needs an id to make the transition to RDF easy. To do that we will use the reserved key "@id". This way, all complex type objects will start with this two keys:

```
{
   "@id": " mythings:Demo1",
   "@type": "ows:ServiceMetadata",
   ...
}
```

11.3.3 Defining type and ids

The key names starting will a "@" are not easy to use in JavaScript. To avoid this we strongly recommend creating synonymous of them. This is the final look of the complex type declaration:

```
"@context": {
   "id": "@id",
   "type": "@type",
   ...
}

"id": " mythings:Demo1",
"type": "ows:ServiceMetadata",
"serviceIdentification":
{
   "id": "mythings:ServiceIdentificationDemo1",
   "type": "ows:ServiceIdentification"
   ...
},
"serviceProvider":
{
   "id": "mythings:serviceProviderDemo1",
   "type": "ows:ServiceProvider"
   ...
},
...
```

11.3.4 Defining data types

JSON-LD does not provide this capability and we need to rely in additional languages to do this such as OWL or JSON Schema. This document already provides the recommendation to do more work on this direction to determine the best approach.

11.3.4.1 Defining enumerations

JSON-LD does not provide this capability but it could be done with JSON Schema.

11.3.4.2 Inheritance and subclassing

Current version on JSON Schema does not seem to support inheritance or subclassing. Only using OWL we can express inheritance.

11.4 Geospatial data types

As we said before, geospatial data types can be expressed as using WKT notation.

11.5 Sharing the @context with several instances

Even if this was not illustrated before in this document the @context part of a JSON-LD file can be stored in an independent file and included and shared by more than one instance. This way is @context document part could be storied in the schemas.opengis.net and imported for each instance.

12 Recommendations

This is the list of recommendations exposed before and collected here as a reference:

Recommendation 1: Consider extending JSON schema to fully describe the properties of a feature type, including units in alphanumeric properties and CRS in the geometric attributes instead of having to repeat them in each instance. Target: OWS Common

Recommendation 2: Consider the possibility that OGC assists the IETF team in moving the JSON Schema forward. Target: Architecture.DWG and OWS Common

Recommendation 3: Consider the possibility that OGC defines specific types for OGC/SIO geometry types. Target: Architecture.DWG and OWS Common

Recommendation 4: Consider the combined use of JSON schema and the @context section of a JSON-LD file (possibly in combination with the ontologies linked to it) as a means for validating a JSON file in the OGC. The next OGC Testbed could include a test on this approach as an activity. Target: Testbed-12

Recommendation 5: Consider the possibilities of using the namespace URIs in @context section of a JSON-LD file as a means to connect to formal ontologies structured in OWL SKOS or other RDF encoding as a way to validate complex types in JSON files in the OGC. The next OGC Testbed could include a test on this approach as an activity. Target: Testbed-12

Recommendation 6: Consider TopoJSON as a model to create a JSON encoding that is different (not just an extension, because addresses a topic that GeoJSON can not consider) but can be mapped and automatically converted into a GeoJSON file (using for example a JavaScript library). Target: OWS Common

Recommendation 7: Connect work in previous testbeds about a WPS profile for topological applications with the TopoJSON to study the applicability and interoperability of TopoJSON in OGC standards such as WPS and WFS. Target: Testbed 12

Recommendation 8: Produce an OGC best practice for converting XML documents into JSON based on OGC 14-009r1 and some other considerations exposed in this ER. Target: OWS Common

Recommendation 9: Include adding "@type" keys to JSON objects as a good practice to makethe transition to JSON-LD and RDF easier. It is also good practice that type names are qualified with a abbreviated namespaces (e.g.: ows:ServiceIdentification) that could be later dereferenced using JSON-LD @context. Target: OWS Common with OAB

Recommendation 10: Include in a best practice for JSON a subclause for linking to other objects in JSON, using the natural approaches that JSON-LD provides for both simple links and atom links. Target: OWS Common.SWG

Recommendation 11: Include in the JSON best practice that if a fragment of a XML document contains a geospatial object then when converting to JSON, consider using the GeoJSON equivalent type. Target: OWS.Common

Recommendation 12: Adopt the creation of specific JSON schema documents as a means of defining feature types and as a means for feature instance validation (as the equivalent of GML application schema). Target: WFS and OAB

Recommendation 13: Consider carefully the unsolved issue where GeoJSON coordinates prevents a natural way to apply JSON-LD to GeoJSON and an automatic conversion to RDF. Following recommendations are proposing alternative solutions. Target: OAB

Recommendation 14: Respect the original format of GeoJSON and apply a piece of code to transform GeoJSON into WKT JSON for simple features on the fly to obtain RDF notation from GeoJSON. Target: OAB

Recommendation 15: Consider JSON-LD as an alternative for creating GML application schemas as a means of defining feature types and as a mean for validation. Target: OWS Common.SWG and OAB

Recommendation 16: Add a BBOX element in the WKT standard. Target: CR to Simple Features for SQL

Recommendation 17: Distribute this JSON schema, and the examples validated with it, in schemas.opengis.net when OWS context JSON standard gets approved. Target: OWS Context

Recommendation 18: Distribute this JSON-LD @context, and the examples validated with it, in schemas.opengis.net when OWS context JSON standard gets approved. Target: OWS Common.SWG and OAB

Recommendation 19: Create an extension of OWS Context JSON for illustrating how to reference GeoJSON data both embedded or linked. Target: CR to OWS Context.SWG

Recommendation 20: Consider the HTML Microdata approach as a new encoding for OWS Context as a way to improve auto-discovery of OGC services and geospatial resources. Target: OWS Context

Recommendation 21: Consider the benefits and drawbacks of extending schema.org vocabularies with a new type for "geospatial resource" that completely matches with OWS Context conceptual model. Target: OWS Context

Recommendation 22: Consider JSON-LD encodings for HTML structured data to create another encoding for OWS Context. Target: OWS Context

Recommendation 23: Elaborate a converge JSON as a new standard encoding for GMLCov. Target: WCS.SWG

Recommendation 24: Include in OWS Common recommendations on how to provide service metadata in JSON derived from the UML models. Include as part of the OWS Common Schemas @context documents for independent validation of the 4 main

sections of the Service Metadata. A JSON schema document can also be provided. Target: OWS Common

Recommendation 25: Consider OWS Context as the *Contents* section of service metadata document. In particular adopt the OWS Context JSON encoding in the JSON encoding of OWS Common. Target: OWS Common

Recommendation 26: OWS Common should recommend a REST requests for GetCapabilities and should recommend JSON as a default request. Target: OWS Common

Recommendation 27: Define the OWS Common minimum set of parameters in a request as a @context fragment. A JSON schema can also be provided. Target: OWS Common

Recommendation 28: In WMS 1.4 include a encoding for GetFeatureInfo responses based on GeoJSON but replacing the geometry part by maker of the position of the query and the position of the returned feature. If returned objects correspond to simple features, return an "id" that allows recovering the geometry using an additional WFS query. Target: WMS.SWG

Recommendation 29: Consider to include tileJSON in a non normative example for the WMTS Simple profile. Target: WMS.SWG

Recommendation 30: Include in the WMTS Simple profile the JSON description of the WMTS simple two tile matrix sets. Target: WMS.SWG

Recommendation 31: For a WFS serving GeoJSON, force the features to have a property that contains the feature type. Target: WFS.SWG

Recommendation 32: Consider that WFS DescribeFeatureType returns a @context section describing the Feature type in GeoJSON. Target: WFS.SWG

Recommendation 33: Consider using either JSON Pointer or JSON Path in places where a XPath is required. Target: WFS.SWG

Recommendation 34: Consider the schema.org model for describing a geospatial user feedback model Target: GUF.SWG

Recommendation 35: Consider a JSON-LD encoding for Geospatial User Feedback that maps the conceptual model to the http://schema.org/AggregateRating and http://schema.org/Review elements Target: GUF.SWG

Recommendation 36: Consider including experimentation in GUF and OWS Context using microdata format in JSON-LD for geospatial features. Target: Testbed-12

13 Future work

The previous section enumerates the recommendations justified throughout this document. Each recommendation specifies that target group for future work. The implementation of them needs a coordinated effort between OGC TC WGs and for next Testbeds. This section provides a short list of ways of moving forward.

13.1 Future work for the TC WGs

This ER elaborated in the Testbed 11 proposes a set of recommendations and many examples on how to move forward in the use of JSON in OGC standards. The first general recommendation for the TC is to propose a way forward in the use of JSON and JSON-LD. OGC TC needs to decide on:

- The way geometries need to be encoded in JSON.
- The exact rules to encode UML models into JSON-LD and eventually to translate XML encoding into JSON encodings.
- The way to validate JSON/JSON-LD and how to distribute validation schemas in the web (e.g. schemas.opengis.net)
- The general adoption of JSON-LD (or only JSON).
- A way to encode links in JSON.
- How to encode GetCapabilities in JSON-LD
- Add BBOX to WKT standard.

This document contains specific recommendations on concrete aspects that OGC TC could take into consideration.

13.2 Future work for next Testbeds

This ER elaborated in the Testbed 11 proposes a set of recommendations for the use of JSON and JSON-LD in OGC services. Some practical developments have been conducted but no extensive testing was done. As a general recommendation, there is a need for more experimentation in JSON-LD and OGC standards. Our recommendation is that the OGC Testbed 12 needs to include a general thread on JSON-LD services and data formats where several services (e.g. WFS, WPS, WCS etc) provided by several participants use JSON-LD encodings extensively for ServiceMetadata, data formats, POST requests etc. Service chain and integration in clients need to be demonstrated.

In addition OGC Testbed 12 could explore the use of HTML and JSON encodings for microformats and schema.org as a way to improve discoverability of services and geospatial data in the web. In particular, OWS Context and Geospatial User Feedback standards can benefit from this approach.

13.3 GeoJSON in W3C Prov

JSON-LD was been proposed as an alternative encoding for W3C Prov by University of Southampton, UK, edited by Trung Dong Huynh, Michael O. Jewell, Amir Sezavar Kshavarz, Danius T. Michaelides, Huanjia Yang, and Luc Moreau (http://www.w3.org/Submission/2013/01/Comment/). Some similar work can be seen here: https://gist.github.com/stain/6027751. An interesting line of work could be to try to use prov-json in conjunction with GeoJSON.

Annex A

Use cases (informative)

A.1 General

This annex presents the use case that was included in the Testbed 11 scenarios.

A.2 Integrating GeoJSON into the semantic web

In this use case, we have two services that are able to serve the same information using two different data models. The information served is data about incidents in San Frencisco:

- ImageMatters SPARQL service is serving the data using a triple store and returns an RDF encoded in N3 (and TTL, RDF/XML etc).
- Cubewebx WFS service is serving it using a data model based on geopatial entities expressed in GeoJSON (and GML).

The objective is have an automatic way to go from GeoJSON encoding to some equivalent representation in RDF.

Image matters request is:
http://ows.usersmarts.com/ldapp/ows11/demo/ems/sfpd/incidents/11615608228150.n3
and returns:

```
@prefix hswg:
<http://www.opengis.net/testbed11/ont/incident/hswg#> .
@prefix geosparql: <http://www.opengis.net/ont/geosparql#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix wgs84: <http://www.w3.org/2003/01/geo/wgs84_pos#> .

<http://ows.usersmarts.com/ldapp/ows11/demo/ems/sfpd/incidents/11
615608228150>
   a hswg:HSWGIncident ;
   hswg:hasAddress      [
         hswg:Address ;
         hswg:city            "San Francisco" ;
         hswg:fullAddress     "0 Block of HARDING RD" ;
         hswg:policeDistrict  "TARAVAL" ;
         hswg:state           "CA"
   ] ;
   hswg:incidentDate    "2011-12-10"^^xsd:date ;
```



```
   hswg:incidentNumber  "116156082" ;
   hswg:incidentTime    "09:10:00"^^xsd:time ;
   hswg:incidentType
<http://www.fgdc.gov/HSWG/taxonomy/IncidentTypes#CivilRioting> ;
   hswg:location [
         a wgs84:Point , geosparql:Point ;
         geosparql:asWKT  "POINT (-122.4977043
37.72473844)"^^geosparql:wktLiteral ;
         wgs84:lat        37.72473844 ;
         wgs84:long       -122.4977043
   ] ;
   hswg:resolution      "NONE" ;
   hswg:summary         "MALICIOUS MISCHIEF, VANDALISM" .
```

Cubewebx request is:
http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/HSWG_Incidents?f=application/json and it returns:

```
{
   "type": "FeatureCollection",
   "features":
   [
      {
         "type": "Feature",
         "geometry":
         {
            "type": "Point",
            "coordinates":
            [
               -122.4977043,
               37.72473844
            ]
         },
         "properties":
         {
            "gmlid":
"CWFID.HSWG_INCIDENTS.0.0.4D91EC1FAEC21F639C2324020000",
            "IncidntNum": "116156082",
            "Category": "VANDALISM",
            "Descript": "MALICIOUS MISCHIEF, VANDALISM",
            "HSWG_Category": "Civil Rioting",
            "DayOfWeek": "Saturday",
            "Date": "2011-12-10",
            "Time": "09:10:00",
            "PdDistrict": "TARAVAL",
            "Resolution": "NONE",
            "Address": "0 Block of HARDING RD",
            "Location": "(37.7247384376425, -122.49770432371)",
```

```
                "PdId": "11615608228150"
            }
        },
        {
            "type": "Feature",
            ...
        }
    ]
}
```

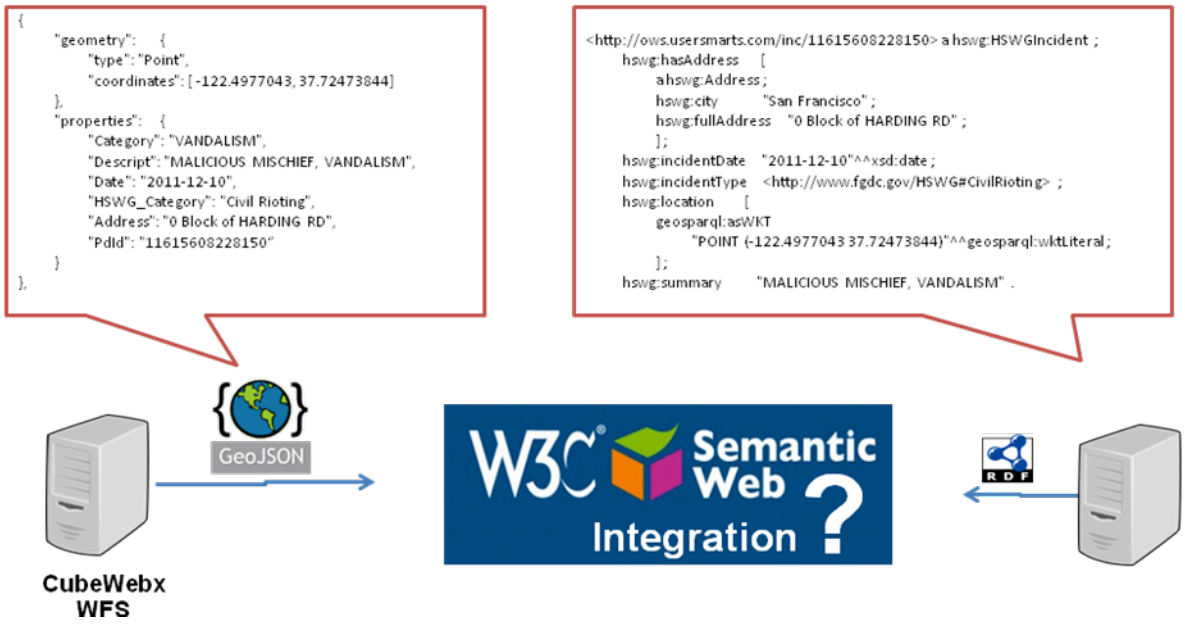

Figure 15: How to achieve the integration of the GeoJSON data into the semantic world?

By adding some identifiers a type and a @context section to the document we are ready to transform the GeoJSON into an RDF format:

```
{
    "@context":
    {
        "hswg":
"http://www.opengis.net/testbed11/ont/incident/hswg/",
        "xsd": "http://www.w3.org/2001/XMLSchema#",
        "rdfs": "http://www.w3.org/2000/01/rdf-schema#",

        "geojson": "http://ld.geojson.org/vocab#",
        "ogc_geo": "http://www.opengis.net/ont/geosparql#",

        "id": "@id",
```

```
      "type": "@type",
      "features": "geojson:features",
      "geometry": "_:",
      "properties": "_:",

      "asWKT": {"@id": "ogc_geo:asWKT", "@type":
"ogc_geo:wktLiteral"},

      "featureType": "geojson:Feature",
      "featureCollectionType": "geojson:FeatureCollection",
      "pointType": "geojson:Point",

      "Feature": "featureType",
      "FeatureCollection": "featureCollectionType",
      "Point": "pointType",

      "IncidntNum": {"@id": "hswg:incidentNumber", "@type":
"xsd:string"},
      "HSWG_Category": {"@id": "hswg:incidentType", "@type":
"xsd:string"},
      "Descript": {"@id": "hswg:summary", "@type": "xsd:string"},
      "Date": {"@id": "hswg:incidentDate", "@type": "xsd:date"},
      "Time": {"@id": "hswg:incidentTime", "@type": "xsd:time"},
      "PdDistrict": {"@id": "hswg:hswg:Address/policeDistrict",
"@type": "xsd:string"},
      "Resolution": {"@id": "hswg:hswg:Address/resolution",
"@type": "xsd:string"},
      "Address": {"@id": "hswg:Address/fullAddress", "@type":
"xsd:string"}
   },

   "id": "hswg:collection1",
   "type": "FeatureCollection",
   "features":
   [
      {
         "id": "hswg:11615608228150",
         "type": "Feature",
         "geometry":
         {
            "id": "hswg:11615608228150",
            "type": "Point",
            "coordinates":
            [
               -122.4977043,
               37.72473844
            ]
         },
         "properties":
         {
            "id": "hswg:11615608228150",
```

```
            "type": "hswg:HSWGIncident",
            "gmlid":
"CWFID.HSWG_INCIDENTS.0.0.4D91EC1FAEC21F639C2324020000",
            "IncidntNum": "116156082",
            "Category": "VANDALISM",
            "Descript": "MALICIOUS MISCHIEF, VANDALISM",
            "HSWG_Category": "Civil Rioting",
            "DayOfWeek": "Saturday",
            "Date": "2011-12-10",
            "Time": "09:10:00",
            "PdDistrict": "TARAVAL",
            "Resolution": "NONE",
            "Address": "0 Block of HARDING RD",
            "Location": "(37.7247384376425, -122.49770432371)",
            "PdId": "11615608228150"
        }
    },
    {
        "id": "hswg:11099249228165",
        "type": "Feature",
        ...
    }
  ]
}
```

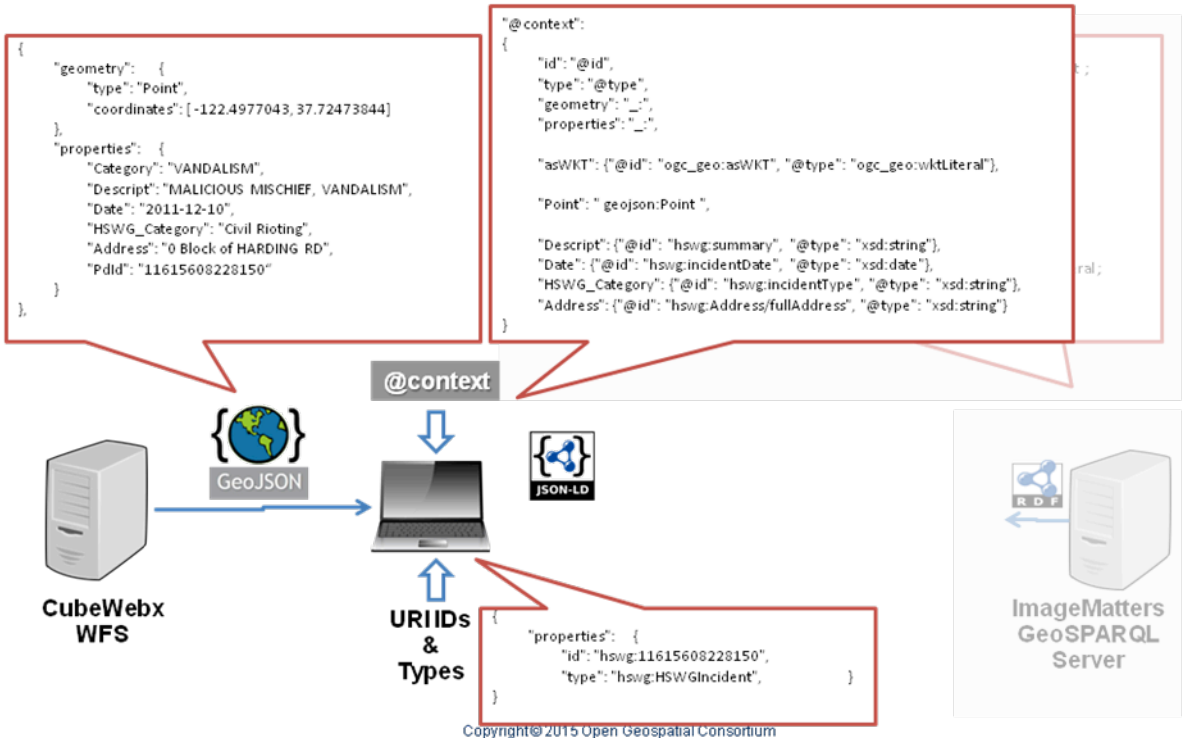

Figure 16: Adding @context @id and @type we convert GeoJSON into JSON-LD

Know we apply a modified JSON-LD parser GeoJSON coordinates array is transformed into Well Known Text, and then, the JSON-LD is transformed into RDF/n3. The process been explained in subclause 7.5 . The result is:

```
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0> <http://www.opengis.net/ont/geosparql#asWKT> "POINT(-
122.4977043
37.72473844)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
.
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0>
<http://www.opengis.net/testbed11/ont/incident/hswg/Address/fullA
ddress> "0 Block of HARDING RD" .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0>
<http://www.opengis.net/testbed11/ont/incident/hswg/hswg:Address/
policeDistrict> "TARAVAL" .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0>
<http://www.opengis.net/testbed11/ont/incident/hswg/hswg:Address/
resolution> "NONE" .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0>
<http://www.opengis.net/testbed11/ont/incident/hswg/incidentDate>
"2011-12-10"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0>
<http://www.opengis.net/testbed11/ont/incident/hswg/incidentNumbe
r> "116156082" .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0>
<http://www.opengis.net/testbed11/ont/incident/hswg/incidentTime>
"09:10:00"^^<http://www.w3.org/2001/XMLSchema#time> .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0>
<http://www.opengis.net/testbed11/ont/incident/hswg/incidentType>
"Civil Rioting" .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0> <http://www.opengis.net/testbed11/ont/incident/hswg/summary>
"MALICIOUS MISCHIEF, VANDALISM" .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://ld.geojson.org/vocab#Feature> .
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.opengis.net/testbed11/ont/incident/hswg/HSWGIncident>
.
```

```
<http://www.opengis.net/testbed11/ont/incident/hswg/collection1>
<http://ld.geojson.org/vocab#features>
<http://www.opengis.net/testbed11/ont/incident/hswg/1161560822815
0> .
<http://www.opengis.net/testbed11/ont/incident/hswg/collection1>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://ld.geojson.org/vocab#FeatureCollection> .
```

We have produced a notation that is now equivalent to the ImageMatters GeoSparql server an achieved integration in the semantic web.

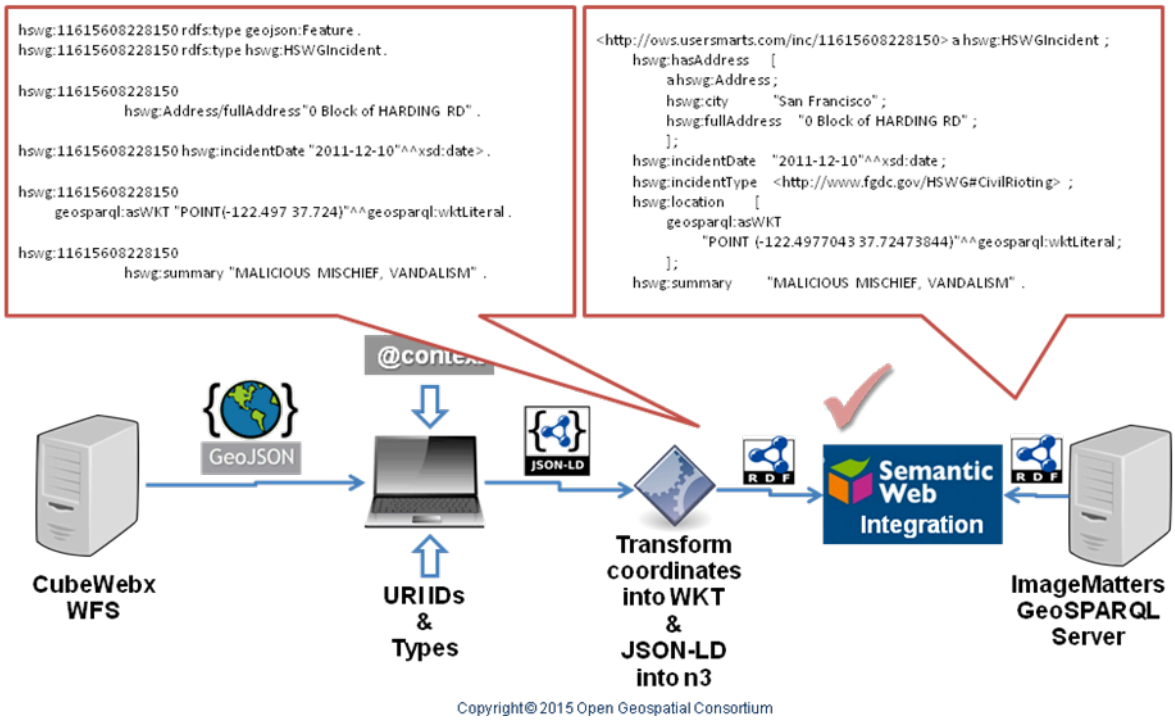

Figure 17: An automatic process can convert JSON-LD into n3 triples. A conversion of GeoJSON coordinates in WKT is needed.

Annex B

JSON Schema validation for OWS Context GeoJSON

B.1 General

GeoJSON encoding for the OWS Context describes the encoding in GeoJSON of the OWS Context Model that is presented in abstract terms in a the OGC 12-080r2 document. The goal of OWS Context has been to allow many types of OGC data delivery services to be referenced and therefore exploited (for example, not just OGC Web Map Service but also OGC Web Feature Service, OGC Web Coverage Service and OGC Web Processing Service) but it does not explicitly define the encoding of these services in the core, only the general approach to be used for different types of service interface.

At the time of writing this ER, OWS Context GeoJSON was in the request for comments phase in the OGC and can be still be modified according with the comments received.

Clause 8.1 describes how GeoJSON encoding for OWS Context can be validated using JSON Schema.

B.2 GeoJSON schema for OWS Context GeoJSON

We present the full version of the schema. This JSON schema will be used to validate the examples that will accompany the final version of the standard. Eventually it could also b the first JSON Schema in schemas.opengis.net.

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "OWS Context JSON schema",
   "type": "object",
   "required": [ "type", "id",  "properties"],
   "properties": {
      "type": { "enum": [ "FeatureCollection" ] },
      "id" : { "type": "string", "format": "uri" },
      "properties": {
         "type": "object",
         "required": [ "links", "lang", "title", "updated" ],
         "properties": {
            "links" : {
               "type": "object",
               "required": [ "profiles"],
               "properties": {
                  "profiles": {
                     "type": "array",
                     "items": { "$ref": "#/definitions/links" }
```

```
                  },
                  "via": {
                     "type": "array",
                     "items": { "$ref": "#/definitions/links" }
                  }
               }
            },
            "lang" : { "type": "string" },
            "title" : { "type": "string" },
            "subtitle" : { "type": "string" },
            "updated" : { "type": "string", "format": "date-time"
},
            "authors" : { "$ref": "#/definitions/authors" },
            "publisher" : { "type": "string" },
            "generator" : {
               "type": "object",
               "properties": {
                  "title" : { "type": "string" },
                  "uri" : { "type": "string", "format": "uri" },
                  "version" : { "type": "string" }
               }
            },
            "display" : {
               "type": "array",
               "items": {
                  "type": "object",
                  "properties": {
                     "pixelWidth": { "type": "number" },
                     "pixelHeight": { "type": "number" },
                     "mmPerPixel": { "type": "number" }
                  }
                }
            },
            "rights" : { "type": "string" },
            "bbox" : {
               "type": "array",
               "minItems" : 4,
               "items": { "type": "number" }
            },
            "date" : { "type": "string" },
            "categories" : { "$ref": "#/definitions/categories" }
         }
      },
      "features" : {
          "type": "array",
          "items": {
            "type": "object",
            "required": [ "type", "id",  "properties"],
            "properties": {
               "id" : { "type": "string", "format": "uri" },
```



```
                "type": { "enum": [ "Feature" ] },
                "geometry": { "$ref": "#/definitions/geometry" },
                "properties": {
                   "type": "object",
                   "required": [ "title", "updated"],
                   "properties": {
                      "title" : { "type": "string" },
                      "abstract" : { "type": "string" },
                      "updated" : { "type": "string", "format":
"date-time" },
                      "authors" : { "$ref":
"#/definitions/authors" },
                      "publisher" : { "type": "string" },
                      "rights" : { "type": "string" },
                      "date" : { "type": "string" },
                      "links" : {
                         "type": "object",
                         "properties": {
                            "previews": {
                               "type": "array",
                               "items": { "$ref":
"#/definitions/links" }
                            },
                            "alternates": {
                               "type": "array",
                               "items": { "$ref":
"#/definitions/links" }
                            },
                            "data": {
                               "type": "array",
                               "items": { "$ref":
"#/definitions/links" }
                            },
                            "via": {
                               "type": "array",
                               "items": { "$ref":
"#/definitions/links" }
                            }
                         }
                      },
                      "date" : { "type": "string" },
                      "active": { "enum": [ true, false ] },
                      "categories" : { "$ref":
"#/definitions/categories" },
                      "minscaledenominator": { "type": "number" },
                      "maxscaledenominator": { "type": "number" },
                      "folder" : { "type": "string" },
                      "offerings" : {
                         "type": "array",
                         "items": {
                            "type": "object",
```

```
                            "required": [ "code"],
                            "properties": {
                               "code": { "type": "string",
"format": "uri" },
                               "operations" : {
                                  "type": "array",
                                  "items" : {
                                     "type": "object",
                                     "required": [ "code",
"method", "href" ],
                                     "properties": {
                                        "code": { "type": "string"
},
                                        "method": { "enum": [
"GET", "POST", "PUT", "DELETE", "HEAD" ] },
                                        "type": { "type": "string"
},
                                        "href": { "type":
"string", "format": "uri" },
                                        "request": { "$ref":
"#/definitions/content" },
                                        "result": { "$ref":
"#/definitions/content" }
                                     }
                                  }
                               },
                               "contents" : {
                                  "type": "array",
                                  "items" : { "$ref":
"#/definitions/content" }
                               },
                               "styles" : {
                                  "type": "array",
                                  "items" : {
                                  "type": "object",
                                  "required": [ "name", "title"],
                                  "properties": {
                                     "name": { "type": "string" },
                                     "title": { "type": "string"
},
                                     "abstract": { "type":
"string" },
                                     "default": { "enum": [ true,
false ] },
                                     "legendURL": { "type":
"string", "format": "uri" },
                                     "content": { "$ref":
"#/definitions/content" }
                                  }
                               }
```



```
                                }
                            }
                        }
                        }
                    }
                }
            }
        }
    },
    "definitions": {
        "links": {
            "title": "links",
            "description": "Properties that all types of links
have. It mimics the Atom link",
            "required": [ "href" ],
            "properties": {
           "href": {"type": "string", "format": "uri" },
           "type" : { "type": "string" },
           "title" : { "type": "string" },
           "lang" : { "type": "string" }
            }
        },
        "authors" :  {
           "title": "authors",
           "description": "Properties that all types of authors
have. It mimics the Atom author",
           "type": "array",
           "items": {
              "type": "object",
              "required": [ "name" ],
              "properties": {
                 "name": { "type": "string" },
                 "email" : { "type": "string", "format": "email" },
                 "uri" : { "type": "string", "format": "uri" }
              }
           }
        },
        "categories" : {
            "title": "categories",
            "type": "array",

            "items": {
              "type": "object",
              "required": [ "term" ],
              "properties": {
                 "term" : { "type": "string"},
                 "scheme" : { "type": "string", "format": "uri"  },
                 "label" : { "type": "string"}
              }
            }
```

```
      },
      "content": {
         "title": "content",
         "type": "object",
         "required": [ "type" ],
         "properties": {
             "type" : { "type": "string"},
             "href" : { "type": "string", "format": "uri"},
             "title" : { "type": "string"},
             "content" : { "type": "string"}
         }
      },
      "geometry": {
         "title": "geometry",
         "type": "object",
         "oneOf": [{
            "properties": {
               "type": { "enum": [ "Point"] },
               "coordinates": {
                  "type": "array",
                  "minItems": 2,
                  "items": { "type": "number"}
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "LineString", "Multipoint"]},
               "coordinates": {
                  "type": "array",
                  "minItems": 2,
                  "items": {
                     "type": "array",
                     "minItems": 2,
                     "items" : { "type": "number" }
                  }
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "Polygon",
"MultiLineString"]},
               "coordinates": {
                  "type": "array",
                  "items": {
                     "type": "array",
                     "minItems": 2,
                     "items": {
                        "type": "array",
```



```
                        "minItems": 2,
                        "items" : { "type": "number" }
                     }
                  }
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "MultiPolygon"] },
               "coordinates": {
                  "type": "array",
                  "items": {
                     "type": "array",
                     "items": {
                        "type": "array",
                        "minItems": 2,
                        "items": {
                           "type": "array",
                           "minItems": 2,
                           "items" : { "type": "number" }
                        }
                     }
                  }
               }
            }
         },
         {
            "properties": {
               "type": { "enum": [ "GeometryCollection"]},
               "geometries": {
                  "type": "array",
                  "items": { "$ref": "#/definitions/geometry" }
               }
            }
         }]
      }
   }
}
```

Annex C

WMTS Simple TileMatrixSet Description in JSON-LD (informative)

C.1 General

The Web Map Tile Service (WMTS) Simple profile defines restrictions that limit the flexibility in implementing a WMTS instance. Adding additional requirements has the goal of simplifying the creation of services and clients. By implementing this profile, clients can more easily combine data coming from different services including from other WMTS instances and even from some tile implementations that are not OGC WMTS based, such as some current distributions of OSM. In fact, most of these tiling services are implicitly following most of the WMTS requirements. Many current WMTS services that implement this profile will have to undergo some changes on how tiles are exposed, and a client that is compatible with WMTS 1.0 will be immediately compatible with this profile. The aim is to align the WMTS standard to other popular tile initiatives which are less flexible but widely adopted.

To do that, two TileMatrixSet are imposed for both World Web Mercator and for CRS84. A TielMatrixSet defines a list of scales and a tiling schema for each of them.

C.2 WMTS Simple profile JSON-LD TileMatrixSet description

The WMTS Simple profile provides the description TileMatrixSet in a table and also as a set of schematron rules. Here, we encode this TileMatrixSets in JSON as an array of two big objects. We hope this could be useful for future implementers of the WMTS Simple profiles clients.

```
{

   "@context":{
      "ows": "http://www.opengis.net/ows/1.1/",
      "wmts": "http://www.opengis.net/wmts/1.0/",
      "wmtss": "http://www.opengis.net/spec/wmts-
simple/1.0/conf/simple-profile/",
      "xsd": "http://www.w3.org/2001/XMLSchema#",

      "id": "@id",
      "type": "@type",

      "tileMatrixSet": "wmts:tileMatrixSet",
      "title": "ows:title",
      "CRSbbox": "ows:BoundingBox",
```



```
      "crs": {"@id":"ows:SupportedCRS", "@type": "@id"},
      "lowerCorner": "ows:lowerCorner",
      "lowerCorner": "ows:lowerCorner",
      "wellKnownScaleSet": {"@id":"wmts:wellKnownScaleSet",
"@type": "@id"},

      "tileMatrix": "wmts:tileMatrix",
      "scaleDenominator": {"@id": "wmts:scaleDenominator",
"@type": "xsd:float"},
      "topLeftCorner": "wmts:topLeftCorner",
      "tileWidth": {"@id": "wmts:tileWidth", "@type":
"xsd:positiveInteger"},
      "tileHeight": {"@id": "wmts:tileHeight", "@type":
"xsd:positiveInteger"},
      "matrixWidth": {"@id": "wmts:matrixWidth", "@type":
"xsd:positiveInteger"},
      "matrixHeight": {"@id": "wmts:matrixHeight", "@type":
"xsd:positiveInteger"}
   },
   "id": "wmtss:",
   "tileMatrixSet": [{
      "type": "wmts:TileMatrixSet",
      "title": "Google Maps Compatible for the World",
      "id": "wmtss:WorldWebMercatorQuad",
      "CRSbbox":
      {
         "type": "ows:CRSbbox",
         "id": "wmtss:WorldWebMercatorQuad/CRSbbox",
         "crs": "http://www.opengis.net/def/crs/EPSG/0/3857",
         "lowerCorner": "POINT(-20037508.3427892, -
20037508.3427892)",
         "upperCorner": "POINT(20037508.3427892,
20037508.3427892)"
      },
      "crs": "http://www.opengis.net/def/crs/EPSG/0/3857",
      "wellKnownScaleSet":
"http://www.opengis.net/def/wkss/OGC/1.0/GoogleMapsCompatible",

      "tileMatrix": [{
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/0",
         "scaleDenominator": 559082264.0287178,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 1,
         "matrixHeight": 1
      },{
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/1",
```

```
         "scaleDenominator": 279541132.0143589,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 2,
         "matrixHeight": 2
      },          {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/2",
         "scaleDenominator": 139770566.0071794,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 4,
         "matrixHeight": 4
      },          {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/3",
         "scaleDenominator": 69885283.00358972,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 8,
         "matrixHeight": 8
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/4",
         "scaleDenominator": 34942641.50179486,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 16,
         "matrixHeight": 16
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/5",
         "scaleDenominator": 17471320.75089743,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 32,
         "matrixHeight": 32
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/6",
```



```
         "scaleDenominator": 8735660.375448715,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 64,
         "matrixHeight": 64
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/7",
         "scaleDenominator": 4367830.187724357,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 128,
         "matrixHeight": 128
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/8",
         "scaleDenominator": 2183915.093862179,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 256,
         "matrixHeight": 256
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/9",
         "scaleDenominator": 1091957.546931089,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 512,
         "matrixHeight": 512
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/10",
         "scaleDenominator": 545978.7734655447,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 1024,
         "matrixHeight": 1024
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/11",
         "scaleDenominator": 272989.3867327723,
```

```
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 2048,
         "matrixHeight": 2048
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/12",
         "scaleDenominator": 136494.6933663862,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 4096,
         "matrixHeight": 4096
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/13",
         "scaleDenominator": 68247.34668319309,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 8196,
         "matrixHeight": 8196
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/14",
         "scaleDenominator": 34123.67334159654,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 16392,
         "matrixHeight": 16392
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/15",
         "scaleDenominator": 17061.83667079827,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 32784,
         "matrixHeight": 32784
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/16",
         "scaleDenominator": 8530.918335399136,
```



```
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 65568,
         "matrixHeight": 65568
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/17",
         "scaleDenominator": 4265.459167699568,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 131136,
         "matrixHeight": 131136
      }, {
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldWebMercatorQuad/18",
         "scaleDenominator": 2132.729583849784,
         "topLeftCorner": "POINT(-20037508.3427892,
20037508.3427892)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 262272,
         "matrixHeight": 262272
      } ]
   },{
      "type": "wmts:TileMatrixSet",
      "title": "CRS84 for the World",
      "id": "wmtss:WorldCRS84Quad",
      "CRSbbox":
      {
         "type": "ows:CRSbbox",
         "id": "wmtss:WorldCRS84Quad/CRSbbox",
         "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
         "lowerCorner": "POINT(-180, -90)",
         "upperCorner": "POINT(180, 90)"
      },
      "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
      "wellKnownScaleSet":
"http://www.opengis.net/def/wkss/OGC/1.0/GoogleCRS84Quad",

      "tileMatrix": [{
         "type": "wmts:TileMatrix",
         "id": "wmtss:WorldCRS84Quad/-1",
         "scaleDenominator": 559082264.0287178,
         "topLeftCorner": "POINT(-180, 90)",
         "tileWidth": 256,
         "tileHeight": 256,
         "matrixWidth": 1,
```

```
      "matrixHeight": 1
    }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/0",
      "scaleDenominator": 279541132.0143589,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 2,
      "matrixHeight": 1
    }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/1",
      "scaleDenominator": 139770566.0071794,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 4,
      "matrixHeight": 2
    }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/2",
      "scaleDenominator": 69885283.00358972,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 8,
      "matrixHeight": 4
    }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/3",
      "scaleDenominator": 34942641.50179486,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 16,
      "matrixHeight": 8
    }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/4",
      "scaleDenominator": 17471320.75089743,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 32,
      "matrixHeight": 16
    }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/5",
      "scaleDenominator": 8735660.375448715,
```



```
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 64,
      "matrixHeight": 32
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/6",
      "scaleDenominator": 4367830.187724357,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 128,
      "matrixHeight": 64
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/7",
      "scaleDenominator": 2183915.093862179,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 256,
      "matrixHeight": 128
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/8",
      "scaleDenominator": 1091957.546931089,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 512,
      "matrixHeight": 256
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/9",
      "scaleDenominator": 545978.7734655447,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 1024,
      "matrixHeight": 512
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/10",
      "scaleDenominator": 272989.3867327723,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 2048,
      "matrixHeight": 1024
   }, {
```

```
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/11",
      "scaleDenominator": 136494.6933663862,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 4096,
      "matrixHeight": 2048
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/12",
      "scaleDenominator": 68247.34668319309,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 8196,
      "matrixHeight": 4096
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/13",
      "scaleDenominator": 34123.67334159654,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 16392,
      "matrixHeight": 8196
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/14",
      "scaleDenominator": 17061.83667079827,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 32784,
      "matrixHeight": 16392
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/15",
      "scaleDenominator": 8530.918335399136,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 65568,
      "matrixHeight": 32784
   }, {
      "type": "wmts:TileMatrix",
      "id": "wmtss:WorldCRS84Quad/16",
      "scaleDenominator": 4265.459167699568,
      "topLeftCorner": "POINT(-180, 90)",
      "tileWidth": 256,
```



```
            "tileHeight": 256,
            "matrixWidth": 131136,
            "matrixHeight": 65568
         }, {
            "type": "wmts:TileMatrix",
            "id": "wmtss:WorldCRS84Quad/17",
            "scaleDenominator": 2132.729583849784,
            "topLeftCorner": "POINT(-180, 90)",
            "tileWidth": 256,
            "tileHeight": 256,
            "matrixWidth": 262272,
            "matrixHeight": 131136
         }, {
            "type": "wmts:TileMatrix",
            "id": "wmtss:WorldCRS84Quad/18",
            "scaleDenominator": 1066.364791924892,
            "topLeftCorner": "POINT(-180, 90)",
            "tileWidth": 256,
            "tileHeight": 256,
            "matrixWidth": 524544,
            "matrixHeight": 262272
         } ]
      } ]
   }
```

Annex D

JSON in C (informative)

D.1 General

Clause 5 describes how to work with JSON in JavaScript. This is ideal for the client side in a web browser. www.json.org provides links to many libraries for reading and writing many JSON files in other languages that can be useful for development in the server side. This annex describes a library in C that we selected to import a GeoJSON file into the author's GIS software called MiraMon and to assess how difficult is to work with JSON in a language different than JavaScript.

D.2 Experience with cJSON

After testing some other alternatives, cJSON demonstrated to be simpler and easiest way to read a JSON file in C language. The library supposes that the programmer is able to upload a json file into a string by itself. Then, loading the string in a structure tree is as simple as using the function `cJSON_Parse(text);` . The function returns a structure that is of `struct cJSON *` type, that represents the first key of the root object. To navigante in the JSON structure, you can use the elements `next` and `prev` to visit the siblings keys and `child` to go into the child elements of a complex element. Depending on the `type` content, `valuestring`, `valueint` and `valuedouble` can be used to recover the value of a simple key. `cJSON_GetObjectItem()` be used to access a key by its name and `cJSON_GetArraySize` and `cJSON_GetArrayItem` are useful to deal with arrays.

One of the advantages of cJSON is that it is only composed by a single open source .c module and a single include .h module. The compilation of the module is not dependent on the compiler you are using and we were able to use it with no problem both in Visual Studio and in Borland C++.

Annex E

Revision history (informative)

Date	Release	Editor	Primary clauses modified	Description
2015-04-03	v 0.1	Joan Masó	all	First draft
2015-05-31	v.0.2	Joan Masó	all	Consolidation of the draft for the Boulder TC meeting presentation for gathering new input. Revisions from Núria Julià (CREAF) and Dave Weslow (NGA) included.
2015-Jun-22	NA	Carl Reed	Various	Prepare for publication

Bibliography

[1] INSPIRE data models
http://inspire.ec.europa.eu/index.cfm/pageid/2/list/datamodels

[2] JSON Schema: interactive and non interactive validation draft-fge-json-schema-validation-00, fge. Galiegue, Ed, K. Zyp, G. Court, http://tools.ietf.org/html/draft-fge-json-schema-validation-00

[3] Lanthaler M, Gutl C. (2012) On Using JSON-LD to Create Evolvable RESTful Services

[4] Microdata (HTML) http://en.wikipedia.org/wiki/Microdata_(HTML)

[5] Microdata: https://html.spec.whatwg.org/multipage/microdata.html

[6] RDF 1.1 N-Quads; A line-based syntax for RDF datasets
http://www.w3.org/TR/n-quads/

[7] OGC 14-113, OGC JSON Position Statement. Internal document.

[8] OGC 12-121, Define XML and JSON schema for a web linking structure based on RFC 5988. Change Request 242

[9] OGC 06-103r4, OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture
http://www.opengeospatial.org/standards/sfa

