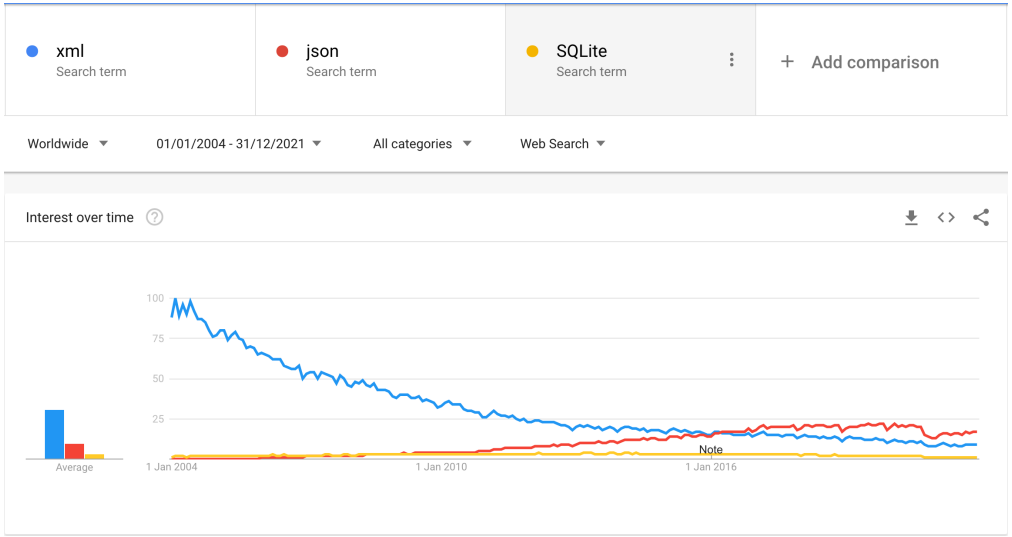
JSON-FG

(OGC Features and Geometries JSON)

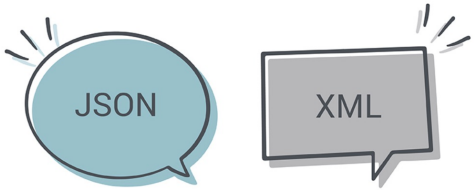
Developers today prefer JSON over XML



APIs

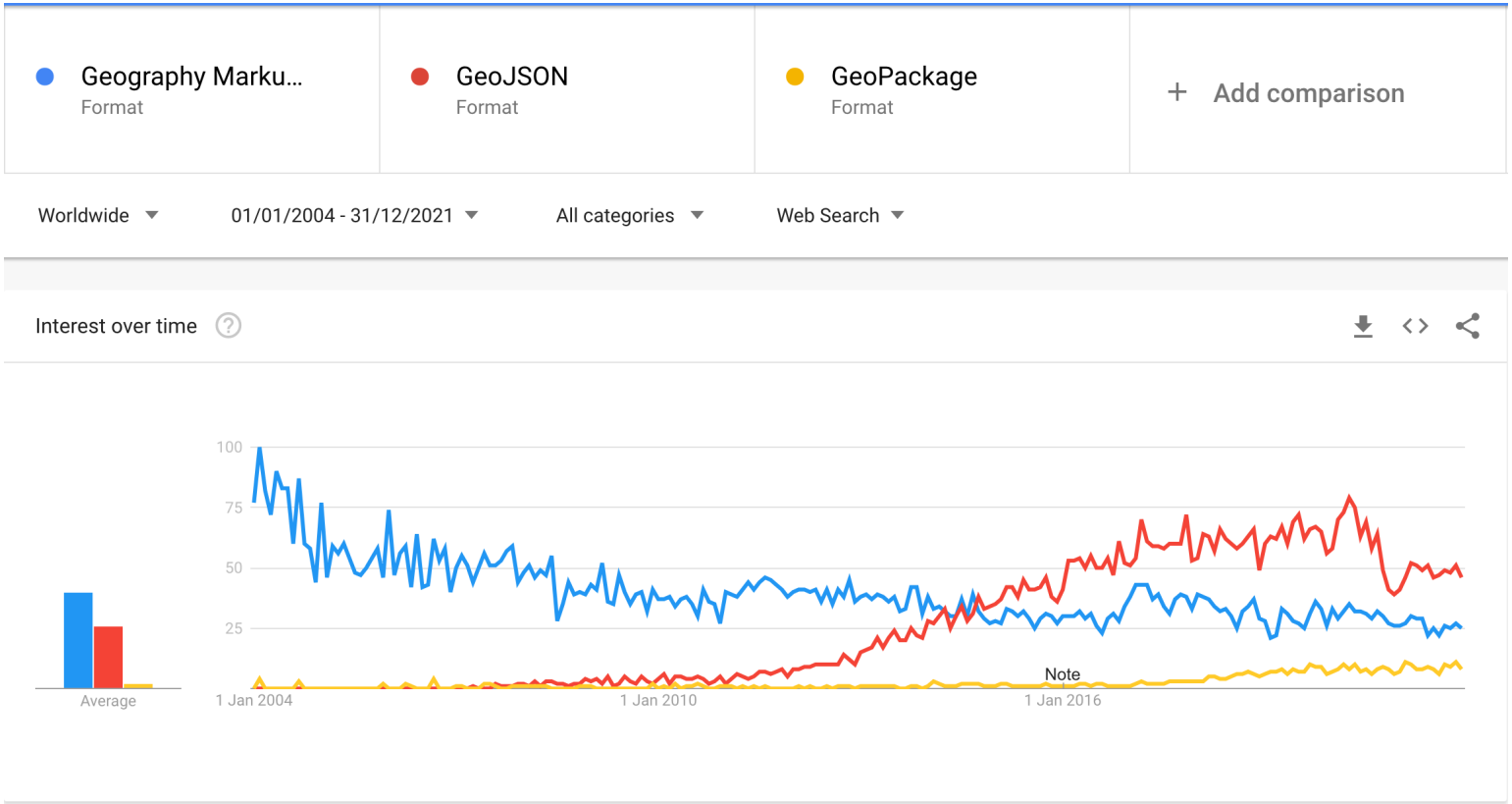
API Formats: Why JSON won over XML

By **Erik Wilde** - November 11, 2020



The vast majority of APIs today are using the *JavaScript Object Notation (JSON)* to represent the structured data that they are exchanging. While JSON has been popular for a number of years now, there still are APIs out there that use the *Extensible Markup Language (XML)* instead, and in some communities, this still is a popular data format.

GeoJSON popular and widely supported, OGC API Features implementations typically support GeoJSON



Motivation for JSON-FG

- Intentional limitations exist in GeoJSON that are an issue for some use cases:
  - Restricted to WGS 84 as Coordinate Reference System
  - Ellipsoidal metrics not supported
  - Points, line strings and polygons – no support for solids or prisms
  - Supports spatial, but not temporal geometries
  - No feature type concept, no information about the schema

Approach

- Develop an OGC Features and Geometries JSON standard addressing the identified limitations
  - Avoid edge cases, focus on capabilities that are useful for many spatial experts
  - Additional capabilities could be added in the future, if there is broad support for the initial OGC Features and Geometries JSON in implementations
- Specify as a superset of GeoJSON
  - That is, valid GeoJSON is also valid OGC Features and Geometries JSON and vice versa
  - Adding additional top-level members and links in the JSON objects (feature and feature collection)
- No dependency on JSON-LD
  - But for those that want to use JSON-LD, avoid conflicts
- It is not the idea to develop a GML-equivalent for JSON

GeoJSON is the starting point

- JSON
  - { ... } is an object with key/value pairs (members)
  - [...] is an array
- GeoJSON
  - A feature collection is an object; predefined keys:
    - "type" – always "FeatureCollection"
    - "features" – an array of features
  - A feature is an object; predefined keys:
    - "type" – always "Feature"
    - "id" – an optional identifier
    - "geometry" – a Simple Feature geometry (Point, LineString, etc.) in WGS 84 longitude, latitude and optional ellipsoidal height
    - "properties" – an object that can contain feature properties, GeoJSON does not place any constraints on the contents

```
{
   "type": "FeatureCollection",
   "features": [
      {
         "type": "Feature",
         "id": "DENW19AL0000giv5BL",
         "geometry": {
            "type": "Point",
            "coordinates": [ 8.7092045, 51.503528 ]
         },
         "properties": {
            "address": "..."
            "lastChange": "2014-04-24T10:50:18Z",
            "built": "2012-03",
            ...
         }
      },
      ...
   ]
}
```

Identifying the feature type(s)

- Features are often categorized by type
  - typically one feature type, but multiple feature types are supported, too
- GIS clients often depend on knowledge about the feature type
  - example: to associate a style to render the feature on a map
- GeoJSON has no concept of feature types or feature schemas

a token for filtering

```
{
   "type": "Feature",
   "id": "DENW19AL0000giv5BL",
   "featureType": "app:building",

   ...

   "links": [
      {
         "href": "https://inspire.ec.europa.eu/
                  featureconcept/Building",
         "rel": "type",
         "title": "This feature is of type 'building'"
      }
   ],
   ...
}
```

in addition, a link to the semantic type definition in some registry, if available

Identifying the schema(s)

- Language: JSON Schema
- Clients can use schemas to validate the JSON document or to derive additional information about the content
- Follow the JSON Schema guidance:
  - *It is RECOMMENDED that instances described by a schema provide a link to a downloadable JSON Schema using the link relation "describedby".*
- Determine that an instance is a GeoJSON / JSON-FG feature though the canonical URIs of the schemas

```
{
   ...,
   "links": [
      {
         "href": "https://ogc-api.nrw.de/lika/v1/
                  collections/gebaeude_bauwerk/schema",
         "rel": "describedby",
         "type": "application/schema+json",
         "title": "JSON Schema of this document"
      },
      {
         "href":"http://schemas.opengis.net/tbd/
                 Feature.json",
         "rel":"describedby",
         "type":"application/schema+json",
         "title":"This document is a JSON-FG Feature"
      },
      {
         "href":"https://geojson.org/schema/
                 Feature.json",
         "rel":"describedby",
         "type":"application/schema+json",
         "title":"This document is a GeoJSON Feature"
      }
   ],
   ...
}
```

links to all schemas that the document conforms to

Encoding temporal information

- GeoJSON supports spatial geometries
- Features are often associated with temporal information, too
- OGC API Features supports not only spatial, but also temporal filtering (`datetime` parameter)
- JSON-FG adds support for the most common case
  - associating a feature with a single temporal instant or interval in the Gregorian calendar
  - main use case is filtering (time slider) or display without the need to understand the feature schema
  - leveraging RFC 3339 and ISO 8601
- No constraints how this primary temporal geometry is derived from the feature properties

```
{
   "type": "Feature",
   ...,
   "time": {
      "interval": [ "2014-04-24T10:50:18Z", ".." ]
   },
   ...,
   "properties": {
      "lastChange": "2014-04-24T10:50:18Z",
      "built": "2012-03",
      ...
   }
}
```

top-level member "time"

Encoding a spatial geometry (1/3)

- GeoJSON supports Simple Features geometries (2D or 2.5D points, line strings, polygons or aggregations of them) in WGS 84
- A geometry that meet these constraints will always be in the "geometry" member from GeoJSON

```
{
   ...,
   "geometry": {
      "type": "Polygon",
      "coordinates": [
         [
            [ 8.709204563652449, 51.50352856284526, 100 ],
...
            [ 8.709204563652449, 51.50352856284526, 100 ]
         ]
      ]
   },
   ...
}
```

Encoding a spatial geometry (2/3)

- Other geometries are added in a top-level member "place"
  - Geometry is a solid or a prism (extruded polygon)
    - Support for arcs and circles under discussion
  - Geometry is in another CRS
  - the CRS is declared in "coordRefSys", also supports ad-hoc compound CRS and coordinate epochs for dynamic CRSs
  - We plan to support a local CRS (cartesian coordinate system with an unknown datum, often used in CAD/BIM)

top-level member "coordRefSys" declares the CRS in the "place" geometry

```
{
   ...,
   "coordRefSys": "http://www.opengis.net/def/crs/
                   EPSG/0/5555",
   "place": {
      "type": "Polyhedron",
      "coordinates": [
         [
            [
               [ 479816.67, 5705861.672, 100 ], ...
               [ 479816.67, 5705861.672, 100 ]
            ]
         ], ...
      ]
   },
   ...
}
```

top-level member "place"

Encoding a spatial geometry (3/3)

- To support GeoJSON readers a fallback geometry can be added in the GeoJSON "geometry" member
- Recommended default for APIs and other JSON-FG generators is to not include a fallback geometry
  - A JSON-FG consumer does not need this information
- Use a media type parameter `compatibility=geojson` for JSON-FG with fallback geometries

```
Accept: application/vnd.ogc.fg+json;compatibility=geojson,
application/vnd.ogc.fg+json; q=0.9, application/geo+json; q=0.8

{
   ...,
   "coordRefSys": "http://www.opengis.net/def/crs/EPSG/0/
                   5555",
   "place": {
      "type": "Polyhedron",
      "coordinates": [
         [
            [
               [ 479816.67, 5705861.672, 100 ], ...
               [ 479816.67, 5705861.672, 100 ]
            ]
         ], ...
      ]
   },
   "geometry": {
      "type": "Polygon",
      "coordinates": [
         [
            [ 8.709204563652449, 51.50352856284526, 100 ], ...
            [ 8.709204563652449, 51.50352856284526, 100 ]
         ]
      ]
   },
   ...
}
```

a valid GeoJSON geometry in "geometry"

Declaring information in the feature collection

- To simplify processing by clients
- For homogenous feature collections, it is sufficient to include the feature type information once – in the feature collection
- If all features in the feature collection have geometries of the same dimension, this can be declared, too
  - 0: points
  - 1: curves
  - 2: surfaces
  - 3: solids
  - no value: unknown or mixed
- Declare a default coordinate reference system

```
{
   "type": "FeatureCollection",
   "featureType": "app:building",
   "geometryDimension": 2,
   "coordRefSys": "http://www.opengis.net/def/crs/
                   EPSG/0/5555",
   "features": [
      ...
   ]
}
```

Relationships and links

- Relationships with other features or other resources like codelists
  - Direct properties of the feature or in embedded JSON objects
- Three patterns for encoding have been identified and are described as guidance, but no plans to specify any requirements
  - Depending on the data and how the data is expected to be used, the preferences of data publishers for one or the other pattern will vary
- Pattern 1: web link in the "links" array
- Pattern 2: like a regular feature property - with a simplified link object
- Pattern 3: like a regular feature property - with a URI value

```
{
   ...,
   "links": [
      {
         "href" : "https://ogc-api.nrw.de/lika/v1/
                   collections/flurstueck/items/
                   05297001600313_____",
         "rel" : "http://www.opengis.net/def/rel/ogc/
                  1.0/within",
         "title" : "Cadastral parcel 313 in district
                    Wünnenberg (016)"
      }
   ],
   "properties": {
      ...,
      "owners": [
         {
            "href": "https://example.org/john-doe",
            "title": "John Doe"
         },
         {
            "href": "https://example.org/jane-doe",
            "title": "Jane Doe"
         }
      ]
   }
}
```

Other topics under discussion

- Declare more Metadata in the feature collection?
  - The zoom level / scale of the geometry (e.g., if the geometry has been simplified)
  - Information that/where geometry has been clipped
  - The Level-of-Detail (LoD) of a feature
  - Metadata to signal to clients/parsers how to process the JSON document (e.g., the media types that the document conforms to)
- Experiments to verify that JSON-FG documents can be used with JSON-LD contexts?

Looking for feedback

- Are these extensions useful for your use cases?
- Are they simple enough to implement?

More Information:

- Draft specification: https://docs.ogc.org/DRAFTS/21-045.html
- GitHub repository: https://github.com/opengeospatial/ogc-feat-geo-json
- Project Board: https://github.com/opengeospatial/ogc-feat-geo-json/projects/1
- Issues: https://github.com/opengeospatial/ogc-feat-geo-json/issues
- Implementations: https://github.com/opengeospatial/ogc-feat-geo-json/tree/main/implementations
- Testbed 17 Engineering Report: http://docs.opengeospatial.org/per/21-017r1.html

JSON-FG Status and Roadmap

- Past Milestones and Activities
  - SWG kick-off: June 1st, 2021
  - Initial proposals for the topics in the charter discussed and agreed
  - Testing in OGC Testbed-17 and the November 2021 Code Sprint
  - Initial internal draft: January 7th, 2022
- Potential Future Milestones
  - First complete draft v0.1: May 2022 ?
    - This is a draft version that we want to keep stable, if possible, for months to support implementations
  - Testing in OGC (e.g. September Code Sprint) and elsewhere
  - Submission of v0.x for OAB Review followed by Public Review: End of 2022 ?
    - Only if there is enough momentum, feedback and implementation support
  - Release of v1.0: 2nd half of 2023 ?
    - Again, only if there is enough momentum, feedback and implementation support

Thank you for your attention!