OGC® DOCUMENT: 22-043R1

External identifier of this OGC® document: http://www.opengis.net/doc/PER/ogc-iso-codesprint2022

OGC®
Making location count.

JOINT OGC AND ISO CODE SPRINT 2022 SUMMARY ENGINEERING REPORT

ENGINEERING REPORT

DRAFT

Submission Date: 2022-10-17
Approval Date: 2022-11-02
Publication Date: YYYY-MM-DD
Editor: Gobe Hobona, Joana Simoes

Note

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.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I EXECUTIVE SUMMARY

Over the past two decades, standards such as ISO 19115:2003 and the OGC Catalogue Services for the Web (CSW) have been integrated into several Spatial Data Infrastructure (SDI) initiatives at national and international levels. These standards leveraged the Extensible Markup Language (XML) which, at the time, was the primary encoding for data exchange across much of Information Technology. In recent times, however, the increasing use of JavaScript Object Notation (JSON) and the uptake of Web Application Programming Interface (API) technologies has meant that modernization of metadata and catalogue approaches is necessary.

In November 2021, OGC and ISO held their first joint code sprint. The success of that first joint code sprint provided the foundation for a second joint code sprint. This ER summarizes the main achievements of the second joint code sprint, conducted between September 14th and 16th, 2022. The second joint code sprint, named the 2022 Joint OGC and ISO Code Sprint — The Metadata Code Sprint, served to accelerate the support of open geospatial standards that relate to geospatial metadata and catalogues. The code sprint was sponsored by Ordnance Survey (OS) at the Gold-level and Geonovum at the Silver-level. The code sprint was held as a hybrid event, with the face-to-face element hosted at the Geovation Hub in London, United Kingdom.

The code sprint focused on the following group of specifications:

- OGC API — Records candidate Standard [1]
- ISO 19115 metadata Standards (i.e., ISO 19115-1, ISO 19115-2, ISO 19115-3)
- OGC Features and Geometries JSON (JSON-FG) candidate Standard [2]
- Spatio-Temporal Asset Catalog (STAC), which leverages the OGC API — Features Standard [3]

The OGC is an international consortium of more than 500 businesses, government agencies, research organizations, and universities driven to make geospatial (location) information and services FAIR — Findable, Accessible, Interoperable, and Reusable. The consortium consists of Standards Working Groups (SWGs) that have responsibility for designing a candidate standard prior to approval as an OGC Standard and for making revisions to an existing OGC Standard. The sprint objectives for the SWGs were to:

- Develop prototype implementations of OGC Standards, including implementations of draft OGC Application Programming Interface (API) Standards;
- Test the prototype implementations;
- Provide feedback to the Editor about what worked and what did not; and
- Provide feedback about the specification document.

Technical Committee 211 (TC 211) of ISO is responsible for the development and publication of standards that relate to geographic information. As with other ISO committees, TC 211 consists

of member nations, as well as liaison partner organizations. TC 211 and OGC have a liaison partnership that enables the organizations to participate in each other's activities and also to collaborate on standards development initiatives. The sprint objectives for ISO/TC 211 were to:

- Support the development of ISO Standards;
- Fix open issues;
- Develop new features; and
- Encourage the implementation of ISO Standards.

This engineering report makes the following recommendations for future innovation work items:

- Initiatives to facilitate implementation of JSON-FG (e.g., three-dimensional (3D) data, cadastral data, etc.);
- Initiatives to facilitate implementation of catalogues; and
- Prototyping of tools for creating metadata (e.g., the automated STAC metadata crawler demonstrated during the sprint).

The engineering report also makes the following recommendations for things that the Standards Working Groups should consider:

- Outreach for promoting JSON-FG;
- Code Sprint for designing profiles of JSON-FG for different communities of interest;
- Documentation of the different roles of catalogues and API, as well as guidance on when to use them;
- Code Sprint on versioning, possibly involving both OGC API — Records and OGC API — Features; and
- Exploring how to move GeoDCAT forward within OGC.

II KEYWORDS

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

hackathon, metadata, code sprint, API, catalogue, record

III SECURITY CONSIDERATIONS

No security considerations have been made for this document.

IV SUBMITTERS

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

NAME	ORGANIZATION	ROLE
Gobe Hobona	Open Geospatial Consortium	Editor
Joana Simoes	Open Geospatial Consortium	Editor
Panagiotis Vretanos	CubeWerx Inc.	Contributor
Núria Julià Selvas	UAB-CREAF	Contributor
Joan Maso	UAB-CREAF	Contributor
Moozhan Shakeri	University of Manchester	Contributor
Clemens Portele	interactive instruments GmbH	Contributor
Matthias Mohr	WWU Münster	Contributor
Jeroen Ticheler	GeoCat	Contributor
Tom Kralidis	Meteorological Service of Canada	Contributor
Byron Cochrane	OpenWork Ltd	Contributor
Andreas Matheus	Secure Dimensions	Contributor
Thijs Brentjens	Geonovum	Contributor

V ABSTRACT

The subject of this Engineering Report (ER) is a code sprint that was held from the 14th to the 16th of September 2022 to advance open standards that relate to geospatial metadata and catalogues. The code sprint was hosted by the Open Geospatial Consortium (OGC) and the International Organization for Standardization (ISO). The code sprint was sponsored by Ordnance Survey (OS) and Geonovum, and held as a hybrid event with the face-to-face element hosted at the Geovation Hub in London, United Kingdom.

1

SCOPE

1 SCOPE

The code sprint described in this engineering report was organized to provide a collaborative environment that enables software developers, users, and architects to work together on open standards that relate to geospatial metadata and catalogues. The engineering report presents the sprint architecture, the results of the prototyping, and a discussion resulting from the prototyping.

A Code Sprint is a collaborative and inclusive event driven by innovative and rapid programming with minimal process and organization constraints to support the development of new applications and open standards. Code Sprints experiment with emerging ideas in the context of geospatial standards, help improve interoperability of existing standards by experimenting with new extensions or profiles, and are used for building proofs of concept to support standards development activities and enhancement of software products.

2

NORMATIVE REFERENCES

2 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Open API Initiative: OpenAPI Specification 3.0.3, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.3.md

Berners-Lee, T., Fielding, R., Masinter, L: IETF RFC 3896, Uniform Resource Identifier (URI): Generic Syntax, https://tools.ietf.org/rfc/rfc3896.txt

ISO: ISO 19115-1, *Geographic information — Metadata — Part 1: Fundamentals*. International Organization for Standardization, Geneva https://www.iso.org/standard/53798.html.

ISO: ISO 19115-2, *Geographic information — Metadata — Part 2: Extensions for acquisition and processing*. International Organization for Standardization, Geneva https://www.iso.org/standard/67039.html.

ISO: ISO/TS 19115-3, *Geographic information — Metadata — Part 3: XML schema implementation for fundamental concepts*. International Organization for Standardization, Geneva https://www.iso.org/standard/32579.html.

3

TERMS, DEFINITIONS AND ABBREVIATED TERMS

3 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

3.1. API

An Application Programming Interface (API) is a standard set of documented and supported functions and procedures that expose the capabilities or data of an operating system, application, or service to other applications [adapted from ISO/IEC TR 13066-2:2016].

3.2. coordinate reference system

A coordinate system that is related to the real world by a datum term name [source: ISO 19111]

3.3. OpenAPI Document

A document (or set of documents) that defines or describes an API. An OpenAPI definition uses and conforms to the OpenAPI Specification (https://www.openapis.org)

3.4. **Metadata**

information about a resource [source: ISO 19115-1:2014, Amendment 2]

3.5. **Web API**

API using an architectural style that is founded on the technologies of the Web [source: OGC API — Features — Part 1: Core]

3.6. Abbreviated terms

API	Application Programming Interface
CRS	Coordinate Reference System
GIS	Geographic Information System
OGC	Open Geospatial Consortium
OWS	OGC Web Services
REST	Representational State Transfer

4

HIGH-LEVEL ARCHITECTURE

4 HIGH-LEVEL ARCHITECTURE

As illustrated in Figure 1, the sprint architecture was designed with the view of enabling client applications to connect to different servers that implement open geospatial standards that relate to metadata and catalogues. Implementations of JSON-FG, ISO 19115, STAC, and OGC API — Records were deployed in participants' own infrastructure in order to build a solution with the architecture shown below in Figure 1.

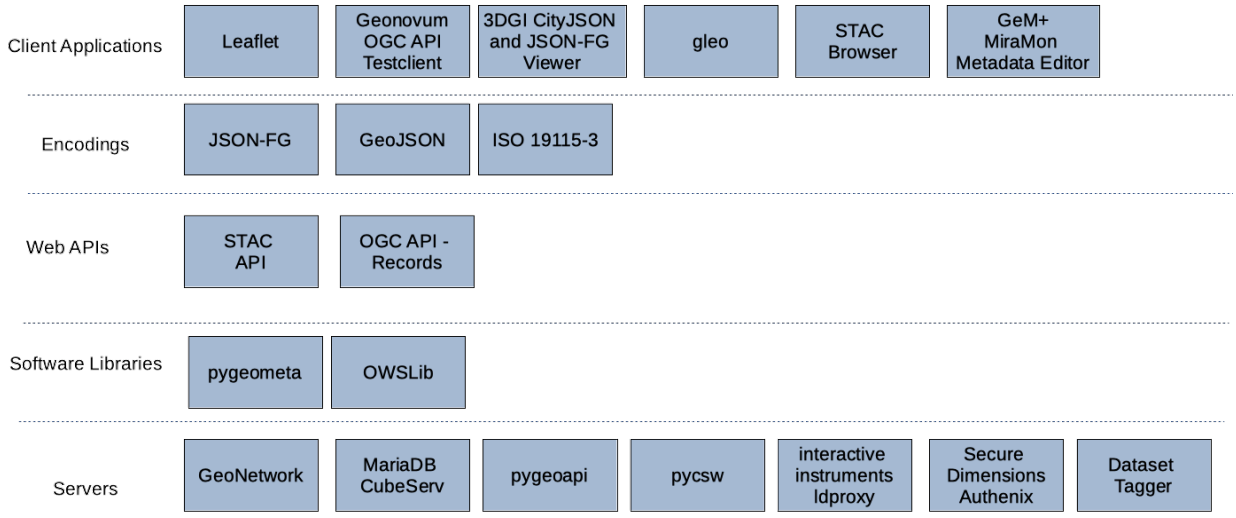

Figure 1 — High Level Overview of the Sprint Architecture

The rest of this section describes the software deployed and standards implemented during the code sprint.

4.1. Approved and Draft Standards

This section describes the approved and draft standards implemented during the code sprint.

4.1.1. OGC API — Records

The OGC API — Records candidate standard provides discovery and access to catalogues of metadata records about resources such as features, coverages, tiles / maps, models, assets, services, or widgets. The candidate standard enables the discovery of geospatial resources by standardizing the way collections of descriptive information about the resources (metadata) are exposed. The candidate standard also enables the discovery and sharing of related resources that may be referenced from geospatial resources or their metadata by standardizing the way all kinds of records are exposed and managed. OGC API — Records can be considered the future successor to the widely implemented Catalogue Services for the Web (CSW) standard.

The candidate standard specifies the information content of a record. A record contains summary descriptive information about a resource that a provider wishes to make discoverable. Records are organized into collections. A record represents resource characteristics that can be presented for evaluation and further processing by both humans and software. Examples of resources include: a data collection, a service, a process, a style, a code list, an Earth observation asset, a machine learning model, a code list, and so on.

4.1.2. JSON-FG

JSON-FG (Features and Geometry JSON) extends GeoJSON to support a limited set of additional capabilities that are out-of-scope for GeoJSON, but that are important for a variety of use cases involving feature data.

The following extensions to the GeoJSON Standard were, in particular, tested during the Code Sprint:

- The ability to use Coordinate Reference Systems (CRSs) other than WGS 84;
- Support for solids as geometry types; and
- The ability to encode temporal characteristics of a feature.

A key design pattern of JSON-FG is that information that can be represented as GeoJSON is encoded as GeoJSON. Additional information is mainly encoded in additional members of the GeoJSON objects. The additional members use keys that do not conflict with GeoJSON or other known GeoJSON extensions. This is done so existing and future GeoJSON clients will continue to parse and understand GeoJSON content. JSON-FG enabled clients will also be able to parse and understand the additional members.

JSON Schema is used to formally specify the JSON-FG syntax.

The draft specification is available at https://docs.ogc.org/DRAFTS/21-045.html.

4.1.3. ISO 19115

ISO 19115 Standards define the schema required for describing geographic information and services by means of metadata. Metadata is information about a resource such as a dataset, web service, or API. This multi-part International Standard is applicable to the cataloguing of datasets, clearinghouse activities, geographic datasets, dataset series, individual geographic features, and feature properties.

The individual parts of ISO 19115 that each serve as an approved standard include:

- ISO 19115-1:2014 defines the schema required for describing geographic information and services by means of metadata;

- ISO 19115-2:2019 extends ISO 19115-1:2014 by defining the schema required for an enhanced description of the acquisition and processing of geographic information, including imagery; and
- ISO/TS 19115-3:2016 defines an integrated XML implementation of ISO 19115-1, ISO 19115-2, and concepts from ISO/TS 19139.

4.1.4. STAC

The SpatioTemporal Asset Catalog (STAC) is a specification that offers a language for describing geospatial information, so it can be worked with, indexed, and discovered. The STAC API offers an interface that implements OGC API — Features. Although STAC has been developed outside of the OGC, in the long term it is envisaged that the STAC API specification will be developed into an OGC Community Standard that implements OGC API building blocks that are relevant for the STAC use cases.

The goals for the STAC part of the code sprint were:

- Alignment with OGC API — Records — Content Model and/or Extensions;
- Alignment with other standards, for example GeoDCAT and/or JSON-LD;
- Collection Search;
- JSON-FG vs. STAC projection extension;
- A possible push towards STAC API 1.0.0 release;
- Advance the STAC ecosystem (e.g., PySTAC, QGIS plugin, ...); and
- Look through the issue trackers, e.g., ecosystem, stac-utils, and stactools-packages.

STAC is a multi-part specification that includes the following constituent parts.

- STAC Item is a representation of a single spatio-temporal asset, encoded as a GeoJSON feature with datetime and links properties.
- STAC Catalog is a JSON-encoded representation of links that provides a structure for organizing and browsing STAC Items.
- STAC Collection extends the STAC Catalog to offer additional information such as the extents, keywords, license, providers, and other elements that describe STAC Items grouped within the Collection.
- STAC API provides a RESTful interface that conforms to the OGC API — Features standard, described in an OpenAPI definition document, and supports search of STAC Items.

Each of the above-listed parts can be used on its own, however the parts have been designed to offer optimal capabilities when used together. An in-depth description of the relationship between the STAC API and Static STAC catalogs is provided in a blog post by Chris Holmes [4].

4.2. Open Source Software Projects

This section describes open source software products that were deployed during the code sprint.

4.2.1. OSGeo GeoNetwork

GeoNetwork is a catalog application for managing spatially referenced resources. It provides metadata editing and search functions as well as an interactive web map viewer.

GeoNetwork is used for (meta)-data management by governments, local communities and private sector. It is also used to discover geospatial (and other) (open) data supporting multiple metadata standards and multiple catalog interfaces.

OGC Standards have been core to the GeoNetwork project and the community is now working on the implementation of the OGC API — Records specification.

4.2.2. Ldproxy

Ldproxy is an implementation of the OGC API family of specifications, inspired by the W3C/OGC Spatial Data on the Web Best Practices. Ldproxy is developed by interactive instruments GmbH, written in Java (Source Code), and is typically deployed using docker (DockerHub). In addition to supporting commonly used data formats for geospatial data, an emphasis is placed on an intuitive HTML representation.

The current version supports PostgreSQL/PostGIS databases, GeoPackages and WFS 2.0 instances as backends for feature data. MBTiles is supported for tilesets.

Ldproxy implements all conformance classes and recommendations of “OGC API — Features — Part 1: Core” and “OGC API — Features — Part 2: Coordinate Reference Systems By Reference” and is an OGC Reference Implementation for the two standards. Ldproxy also supports the OGC API Records draft as well as the draft extensions of OGC API — Features (that is Part 3, CQL2, Part 4 and most of the current proposals discussed by the Features API working group). It supports GeoJSON, JSON-FG, HTML, FlatGeoBuf, CityJSON, glTF 2.0 and GML as feature encodings.

Ldproxy also has implementations for additional resource types: Vector and Map Tiles, Styles, Routes, 3D Tilesets.

Ldproxy is distributed under the Mozilla Public License 2.0.

4.2.3. dataset-tagger

The dataset-tagger open source software product offers an application for fetching metadata from a dataset. The metadata model is derived from that used by OGC API Records.

4.2.4. gleo

The gleo library is an open-source JavaScript library for WebGL-powered geographic maps. The library supports the display of geographical maps and is intended to cover the same use cases as other libraries such as Leaflet, OpenLayers.

4.2.5. OSGeo pygeoapi

pygeoapi is a Python server implementation of the OGC API suite of standards. The project emerged as part of the next generation OGC API efforts in 2018 and provides the capability for organizations to deploy a RESTful OGC API endpoint using OpenAPI, GeoJSON, and HTML. pygeoapi is open source and released under an MIT license. pygeoapi is an official OSGeo Project as well as an OGC Reference Implementation.

pygeoapi supports numerous OGC API Standards. The official documentation provides an overview of all supported standards.

4.2.6. OSGeo pycsw

pycsw is an OGC API — Records and OGC CSW server implementation written in Python. Started in 2010 (more formally announced in 2011), pycsw allows for the publishing and discovery of geospatial metadata via numerous APIs (CSW 2/CSW 3, OpenSearch, OAI-PMH, SRU), providing a standards-based metadata and catalogue component of spatial data infrastructures. pycsw is Open Source, released under an MIT license, and runs on all major platforms (Windows, Linux, Mac OS X). pycsw is an official OSGeo Project as well as an OGC Reference Implementation.

pycsw supports numerous metadata content and API standards, including OGC API — Records — Part 1.0: Core and its associated specifications. The official documentation provides an overview of all supported standards.

4.2.7. OSGeo pygeometa

pygeometa provides a lightweight and Pythonic approach for users to easily create geospatial metadata in standards-based formats using simple configuration files (affectionately called metadata control files (MCF)). The software has minimal dependencies (install is less than 50 kB), and provides a flexible extension mechanism leveraging the Jinja2 templating system. Leveraging the simple but powerful YAML format, pygeometa can generate metadata in numerous standards. Users can also create their own custom metadata formats which can be

plugged into pygeometa for custom metadata format output. pygeometa is open source and released under an MIT license.

For developers, pygeometa provides a Pythonic API that allows developers to tightly couple metadata generation within their systems and integrate nicely into metadata production pipelines.

The project supports various metadata formats out of the box including ISO 19115, the WMO Core Metadata Profile, and the WIGOS Metadata Standard. The project also supports the OGC API — Records core record model as well as STAC (Item).

4.2.8. OSGeo OWSLib

OWSLib is a Python client for OGC Web Services and their related content models. The project is an OSGeo Community project and is released under a BSD 3-Clause License.

OWSLib supports numerous OGC standards, including increasing support for the OGC API suite of standards. The official documentation provides an overview of all supported standards.

4.3. Proprietary products

This section describes proprietary software products that were deployed during the code sprint.

4.3.1. MariaDB CubeWerx CubeServ

The CubeWerx server ("cubeserv") is implemented in C and currently implements the following OGC Standards and draft specifications.

- All conformance classes and recommendations of the OGC API — Features — Part 1: Core Standard.
- Multiple conformance classes and recommendations of the OGC API — Records — Part 1: Core candidate Standard.
- Multiple conformance classes and recommendations of the OGC API — Coverages — Part 1: Core candidate Standard.
- Multiple conformance classes and recommendations of the OGC API — Processes — Part 1: Core Standard.
- Multiple versions of the Web Map Service (WMS), Web Processing Service (WPS), Web Map Tile Service (WMTS) and Web Feature Service (WFS) Standards.
- A number of other "un-adopted" OGC Web Service draft specifications including the Testbed-12 Web Integration Service, OWS-7 Engineering Report — GeoSynchronization Service, and the Web Object Service prototype.

The cubeserv executable supports a wide variety of back ends including Oracle, MariaDB, SHAPE files, etc. It also supports a wide array of service-dependent output formats (e.g., GML, GeoJSON, Mapbox Vector Tiles, MapMP, etc.) and coordinate reference systems.

4.3.2. Geonovum OGC API Testclient

The Geonovum OGC API Testclient is a basic client to showcase the interaction with implementations of OGC API — Features and OGC API — Records. It is built with LeafletJS and jQuery, for simplicity.

4.3.3. 3DGI CityJSON and JSON-FG Viewer

Participants from 3DGI and Geonovum worked on an extension of a CityJSON viewer to enable support for JSON-FG. CityJSON is an OGC Community Standard for a JSON-based encoding for a well-documented subset of the OGC CityGML data model (version 2.0.0). CityJSON defines how to store digital 3D models of cities and landscapes.

4.3.4. GeM+

GeM+ is an application built for the MS Windows environment. It enables users to create, manage and edit metadata from a variety of geographic data. GeM+ is able to read some of the most common formats, and to dynamically transfer to the metadata some information extracted from the data itself. Note that GeM+ is the metadata editor of the MiraMon product suite from UAB-CREAF.

4.4. Other solutions

This section describes other solutions that were deployed for the code sprint.

4.4.1. Secure and Asynchronous Catalogue

There are many factors that influence the response time of a server when it receives a request from a client application. Socket timeouts can occur while the client application waits for a response from the server. To avoid losing the response for requests that require a processing time that is longer than socket timeout, the communication to the client application may continue 'asynchronously.' That means that the service to client application communication does not take part on the original sockets; it happens somehow else.

The proposed solution from the OGC Testbed-18 Secure & Asynchronous Catalogue team was to leverage the HTTP *Prefer* header (as defined in IETF RFC 7240) that enables the client application to indicates to the server about the response preferences. For example, a *Prefer* header with the value "*respond-async*, *wait=10*" indicates that the client application is able

(willing) to wait 10 seconds for a response and that an asynchronous response is preferred. To control a generic carry-on for OGC APIs via asynchronous follow-up communication, the Testbed 18 team recommended definition of OGC specific *Prefer* header parameters. One example is to define the parameter *subscription* which links to an existing subscription that defines a particular way of carrying-on. For example, the following Prefer header extends the previous example indicating the service to use a particular subscription identified via the URI: "*Prefer: respond-async, wait=10, subscription=https://ogc.demo.secure-dimensions.de/sms/subscriptions/4711*". In case the service honors the indicated client application preference, the HTTP response must contain the HTTP Header Preference-Applied including the parameter that was accepted. For the previous example, the acceptance of the subscription option is expressed with the HTTP Response header "*Preference-Accepted: subscription*".

This proposed solution requires however that a generic Subscription Management Service (SMS) exists and that the OGC API implementation introspects the HTTP Request headers and checks for the *Prefer* header. This checking can be achieved in a generic OGC API gateway for example. For the Code Sprint, the Testbed-18 team sought to extend the SMS prototype implemented during Testbed-18 with an additional "carry-on" communication: WebPush as defined by W3C Push API. This required a client and a service implementation. The code sprint provided an opportunity for the Testbed-18 team to explore the proposed solution in collaboration with other sprint participants. The proposed solution leveraged the Authenix product, an OAuth2 / OpenID Connect compliant authorization server based on federated identity management.

4.4.2. University of Manchester Natural Language Processing for Spatial Analysis

Participants from the University of Manchester implemented a Natural Language Processing (NLP) toolkit for conducting spatial analysis. The toolkit takes as input a statement such as "I want to see parks in Westminster" and then displays the locations of the requested features (parks) in the location of interest (Westminster).

5

RESULTS

5 RESULTS

The code sprint included multiple software products and implementations of OGC and ISO Standards. This section presents some of the results from the code sprint.

5.1. gleo

One of the contributors of gleo implemented support for JSON-FG in the code sprint. Support for JSON-FG was implemented to enable anyone to integrate the JSON-FG files into a gleo application.

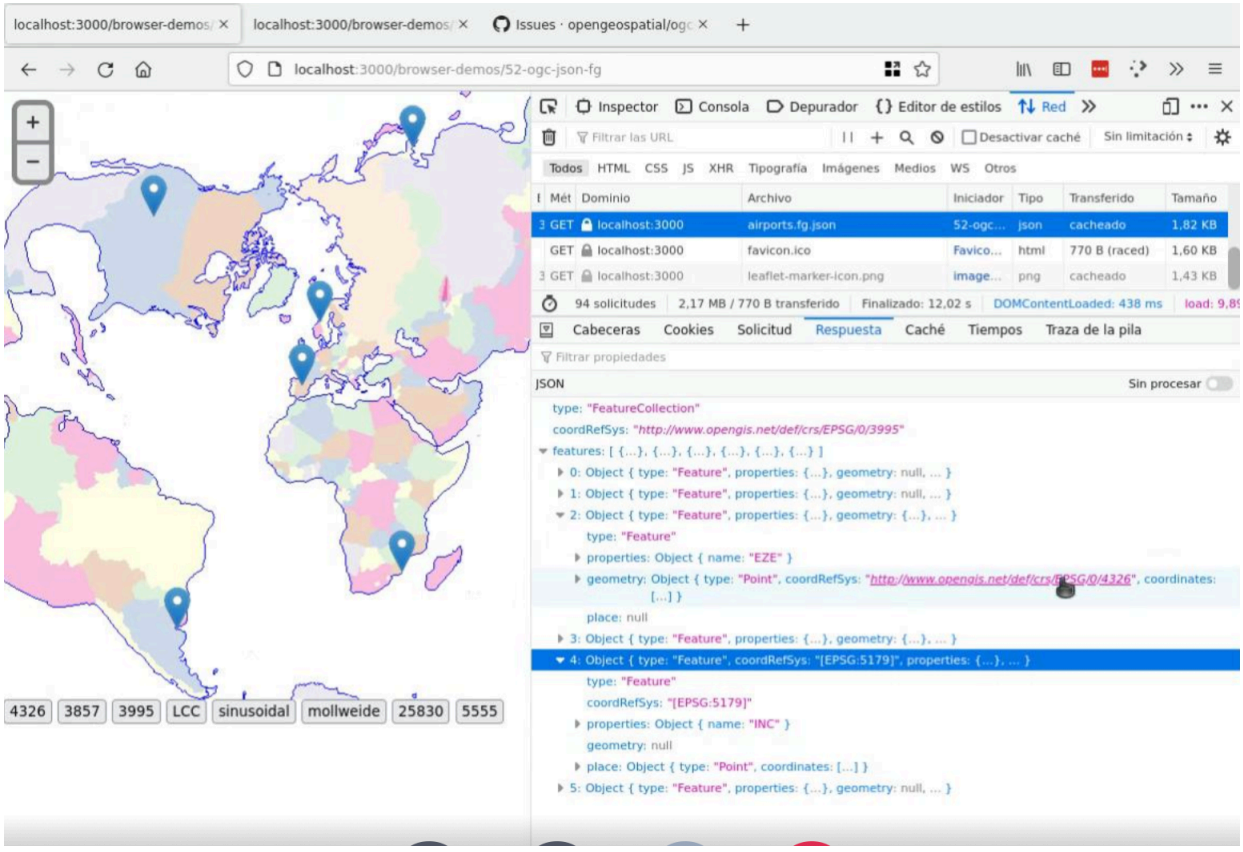

Figure 2 — Screenshot of the gleo demo

5.2. MariaDB CubeWerx CubeSERV

5.2.1. Introduction

CubeWerx has an implementation of a searchable catalogue and during the code spint updated its catalogue to:

- increase interoperability with STAC; and
- develop a harvesting API.

5.2.2. STAC harmonization

By default the CubeWerx catalogue will generate a catalogue record as defined in the DRAFT OGC API — Records — Part 1: Core specification.

During the code sprint, CubeWerx modified the generated record to include the changes discussed in the STAC harmonization lessons. Specifically, previously-defined fields `created` and `updated` were removed from the `properties` section of the record and instead added to the `link` structure.

The following examples illustrate the current structure of a record from the CubeWerx catalogue:


```
    {
      "id": "urn:uuid:00000c8e-d493-11ec-a1bc-43c1b3ebff8b",
      "type": "Feature",
      "created": "2022-09-15T09:17:34Z",
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[ -80.738129, -0.875548 ], [ -78.135155, -0.875548 ],
                         [ -78.135155,  1.346873 ], [ -80.738129,  1.346873 ],
                         [ -80.738129, -0.875548 ], [ -80.738129, -0.875548 ]]]
      },
      "properties": {
        "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct",
        "title": "SENTINEL-1 Product (S1A_IW_GRDH_1SDV_20220111T110053_
20220111T110122_041412_04EC8F_56A0)",
        "crs": "http://www.opengis.net/def/crs/EPSG/0/4326",
        "absoluteOrbitNumber": 41412,
        "missionDataTakeId": "322703",
        "missionId": "S1A",
        "mode": "IW",
        "passDirection": [
          "Descending",
          "Descending"
        ],
        "path": "GRD/2022/1/11/IW/DV/S1A_IW_GRDH_1SDV_20220111T110053_
20220111T110122_041412_04EC8F_56A0",
        "polarization": "DV",
        "processingLevel": 1,
```

```
        "productClass": "standard",
        "productId": "S1A_IW_GRDH_1SDV_20220111T110053_20220111T110122_041412_
04EC8F_56A0",
        "productType": "GRD",
        "productUniqueIdentifier": "56A0",
        "resolutionClass": "high",
        "s3Ingestion": "2022-01-11T14:42:50.970Z",
        "sciHubIngestion": "2022-01-11T14:00:07.747Z",
        "startTime": "2022-01-11T11:00:53Z",
        "stopTime": "2022-01-11T11:01:22Z"
      },
      "links": [
        {
          "href":"s3://sentinel-s1-l1c/GRD/2022/1/11/IW/DV/S1A_IW_GRDH_1SDV_
20220111T110053_20220111T110122_041412_04EC8F_56A0/measurement/iw-vv.tiff",
          "rel": "ogc:data",
          "created": "2022-01-11T11:00:53Z"
        },
        {
          "href":"s3://sentinel-s1-l1c/GRD/2022/1/11/IW/DV/S1A_IW_GRDH_1SDV_
20220111T110053_20220111T110122_041412_04EC8F_56A0/measurement/iw-vh.tiff",
          "rel": "ogc:data",
          "created": "2022-01-11T11:00:53Z"
        },
        {
          "href": "s3://sentinel-s1-l1c/GRD/2022/1/11/IW/DV/S1A_IW_GRDH_1SDV_
20220111T110053_20220111T110122_041412_04EC8F_56A0/preview/map-overlay.kml",
          "rel": "ogc:overlay"
        },
        {
          "href": "s3://sentinel-s1-l1c/GRD/2022/1/11/IW/DV/S1A_IW_GRDH_1SDV_
20220111T110053_20220111T110122_041412_04EC8F_56A0/preview/product-preview.
html",
          "rel": "ogc:product-preview"
        },
        {
          "href": "s3://sentinel-s1-l1c/GRD/2022/1/11/IW/DV/S1A_IW_GRDH_1SDV_
20220111T110053_20220111T110122_041412_04EC8F_56A0/preview/quick-look.png",
          "rel": "ogc:quickview"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv?service=
WRS&version=3.0&catalogueId=sentinelcat&request=GetRepositoryItem&id=urn%3Auuid
%3A00000c8e-d493-11ec-a1bc-43c1b3ebff8b&repoId=3337",
          "rel": "ogc:thumbnail"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/ca
talogues",
          "rel": "service",
          "title": "Link to service ..."
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/ca
talogues/collections/sentinel1cat",
          "rel": "collection",
          "title": "Link to collection..."
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=json&s=ogcapirecord",
          "rel": "self",
```

```
          "title": "Link to this record..."
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=xml&s=ogcapirecord",
          "rel": "alternate",
          "type": "application/xml",
          "title": "Link to record as \"OGC API Record\" encoded as
application/xml"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=html&s=ogcapirecord",
          "rel": "alternate",
          "type": "text/html",
          "title": "Link to record as \"OGC API Record\" encoded as text/html"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=xml&s=cswrecord30",
          "rel": "alternate",
          "type": "application/xml",
          "title": "Link to record as \"CSW Record v3.0\" encoded as
application/xml"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=xml&s=cswrecord20",
          "rel": "alternate",
          "type": "application/xml",
          "title": "Link to record as \"CSW Record v2.X\" encoded as
application/xml"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=xml&s=ebrim",
          "rel": "alternate",
          "type": "application/xml",
          "title": "Link to record as \"ebRIM\" encoded as application/xml"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=xml&s=atom",
          "rel": "alternate",
          "type": "application/xml",
          "title": "Link to record as \"ATOM\" encoded as application/xml"
        },
        {
          "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/c
atalogues/collections/sentinel1cat/items/urn%3Auuid%3A00000c8e-d493-11ec-a1bc-
43c1b3ebff8b?f=xml&s=rss",
          "rel": "alternate",
          "type": "application/xml",
          "title": "Link to record as \"RSS\" encoded as application/xml"
        }
      ]
```

```
}
```

Notice that the previously defined `recordCreated` member is now simply called `created` (with a value of "2022-09-15T09:17:34Z"). The previously defined `created` property has now been moved to the links with `rel=ogc:data` that point to the data products described by the record.

NOTE: The CubeWerx server uses CURIES (`ogc:data`, `ogc:overlay`, `ogc:product-preview`, `ogc:quickview`, `ogc:thumbnail`) for link relations that closely match the file relationships that exist within a Sentinel S1A product. However, a set of link relations relevant to Earth observation imagery should be defined within the OGC.

5.2.3. Harvest API

5.2.3.1. Introduction

The natural way of populating a catalogue through the API would be to use the OGC API — Features — Part 4: Create, Replace, Update and Delete specification. This approach however puts a heavy burden on the client to know exactly how to map metadata about the incoming resource into a catalogue record.

The harvest API, as described in this section, attempts to shift this burden from the client to the server. Rather than the client having to read the resource to be described in the catalogue and figure out how to map its metadata to a catalogue record, the server reads the resource and creates (or updates) one or more new records.

Consider the example of a Sentinel S1A imagery product. Using Part 4 a client would need to scan the Sentinel product directory, figure out which files contain the relevant metadata, etc and then map that metadata to one or more catalogue records. Using the harvest API, the client simply provides the server with a reference to the resource and the server does the rest of the heavy lifting required to create a record(s) that describes the resource.

5.2.3.2. API definition

The harvest API defined during the code sprint defines the following endpoint:

Table 1 — Harvest interface defined during the code sprint

ENDPOINT	METI	DESCRIPTION	BODY OR RESPONSE
/collections/{catalogueId}/harvest	GET	Get a list of harvest requests.	Response is an array of harvest requests.
	PUT	Modify a harvest request.	Body contains a harvest request.
	POS1	Submit a harvest request.	Body contains a harvest request,

ENDPOINT	METHOD	DESCRIPTION	BODY OR RESPONSE
			response is a brief record.
	DELETE	Delete a harvest request	

The harvest API reuses the `/jobs/{jobID}` endpoint defined in the OGC API — Processes — Part 1: Core to provide the status on an asynchronously executing harvest request. The following table summarizes the behavior:

Table 2 — Jobs interface defined during the code sprint

ENDPOINT	METHOD	DESCRIPTION	BODY OR RESPONSE
/jobs/{jobID}	GET	Get the status of a harvest job.	Response is status Info.yaml.
	DELETE	Cancel/remove a harvest job.	Body is empty.

5.2.3.3. Harvest request parameters

The following table defines the parameters of a harvest request:

Table 3 — The parameters of a harvest request

PARAMETER	REQUIREMENT	TYPE	DESCRIPTION
sources	Mandatory	URI	Reference(s) to the resource(s) to be harvested.
harvestInterval	Optional	Period/Interval	Reharvest interval
responseHandlers	Optional	URI	List of callbacks for asynchronous notifications.
links	Optional	URI	Links to related information.

The following skeleton JSON document illustrates a JSON harvest request.

```
{
  "harvestInterval": "P1W",
  "sources": [ {...link...}, {...link...}, ...],
  "responseHandlers": [ "...", "...", ... ]
  "links": [ ... ],
}
```

NOTE 1: The `responseHandlers` parameter was inherited from CSW 3.0. It might be better to define a new OGC header for specifying response handlers OR adding a custom parameter to the Prefer HTTP header.

NOTE 2: A re-harvest based on the `harvestInterval` should make use of conditional headers to only reharvest if something has changed (i.e., use `Last-Modified`, `If-Modified-Since`).

5.2.3.4. Examples

```
CLIENT                                                                SERVER
   |                                                                    |
   |  POST /collections/{collectionId}/harvest  HTTP/1.1                |
   |  Host: www.some-cat-server.com                                     |
   |  Content-Type: application/json                                    |
   |  Prefer: respond-async, wait=10, return=minimal                    |
   |                                                                    |
   |  {                                                                 |
   |     "harvestInterval": "P1W",                                      |
   |     "sources": [                                                   |
   |       {                                                            |
   |         "href": "s3://sentinel-s2-l1c/products/2022/9/15/.../",    |
   |         "type": "..."                                              |
   |       }                                                            |
   |     ],                                                             |
   |     "responseHanders": [                                           |
   |        "mailto:pvretano@pvretano.com",                             |
   |        "http://www.some-web-hook.com"                              |
   |     ]                                                              |
   |  }                                                                 |
   |=================================================================>|
   |                                                                    |
   |  HTTP/1.1 200 OK                                                   |
   |  Content-Type: application/json                                    |
   |  Location: </collections/{collectionId}/harvest/hid007>            |
   |                                                                    |
   |  {                                                                 |
   |    "type": "FeatureCollection",                                    |
   |    .                                                               |
   |    .                                                               |
   |    .                                                               |
   |    "features": [                                                   |
   |      {                                                             |
   |        "id": "urn:uuid:00029e0c-d530-11ec-8f05-cbee6233d299",      |
   |        "type": "Feature",                                          |
   |        "geometry": { ... },                                        |
   |        "properties": {                                             |
   |          "type": "data-product",                                   |
   |          "title": "..."                                            |
   |        },                                                          |
   |        "links": [                                                  |
   |          {                                                         |
   |            "href": "http://...",                                   |
   |            "rel": "self"                                           |
   |          }                                                         |
   |        ]                                                           |
   |      },                                                            |
```

```
|      .                                                          |
|      .                                                          |
|      .                                                          |
|    ]                                                            |
|  }                                                              |
|<================================================================|
```

Figure 3 — Example 1: harvest a resource (sync response)

NOTE 1: A harvest may result in more than one record being created and that is why the response is a collection.

```
CLIENT                                                      SERVER
   |                                                            |
   |  POST /collections/{collectionId}/harvest  HTTP/1.1        |
   |  Host: www.some-cat-server.com                             |
   |  Content-Type: application/json                            |
   |  Prefer: respond-async, wait=10, return=minimal            |
   |                                                            |
   |  {                                                         |
   |     "harvestInterval": "P1W",                              |
   |     "sources": [                                           |
   |       {                                                    |
   |         "href": "s3://sentinel-s2-l1c/products/2022/9/15/.../",
   |         "type": "..."                                      |
   |       }                                                    |
   |     ],                                                     |
   |     "responseHanders": [                                   |
   |        "mailto:pvretano@pvretano.com",                     |
   |        "http://www.some-web-hook.com"                      |
   |     ]                                                      |
   |  }                                                         |
   |=========================================================== >|
   |                                                            |
   |  HTTP/1.1 202 Accepted                                     |
   |  Content-Type: application/json                            |
   |  Prefer-Accept: ...                                        |
   |                                                            |
   |  Location: </collections/{collectionId}/harvest/hid007>    |
   |  Link: <jobs/1013>, rel=monitor                            |
   |<===========================================================|
```

Figure 4 — Example 2: Harvest a resource (async response)

```
CLIENT                                                      SERVER
   |                                                            |
   |  GET /jobs/1013                                            |
   |  Host: www.some-cat-server.com                             |
   |=========================================================== >|
   |                                                            |
   |  HTTP/1.1 200 OK                                           |
   |  Content-Type: application/json                            |
   |                                                            |
   |  {                                                         |
   |    "jobId": "1013",                                        |
   |    "status": "running",                                    |
   |    "type": "harvest",                                      |
   |    "created": "2022-09-15T06:10:00Z"                       |
```

```
|    "links": [                                                  |
|       {"rel": "monitor", "href": /jobs/1013"}                  |
|    ]                                                           |
|  }                                                             |
|<===============================================================|
```

Figure 5 — Example 3: Get the current status of a harvest job

```
CLIENT                                                    SERVER
  |                                                          |
  |  GET /collections/{catalogueId}/harvest                  |
  |  Host: www.some-cat-server.com                           |
  |=========================================================>|
  |                                                          |
  |  HTTP/1.1 200 OK                                         |
  |  Content-Type: application/json                          |
  |                                                          |
  |  [                                                       |
  |    {                                                     |
  |      "id": "hid007",                                     |
  |      "harvestInterval": "P1W",                           |
  |      "sources": [                                        |
  |        {                                                 |
  |          "href": "s3://sentinel-s2-l1c/products/2022/9/15/.../",
  |          "type": "..."                                   |
  |        },                                                |
  |      ],                                                  |
  |      "responseHanders": [                                |
  |        "mailto:pvretano@pvretano.com",                   |
  |        "http://www.some-web-hook.com"                    |
  |      ]                                                   |
  |    }                                                     |
  |  ]                                                       |
  |<=========================================================|
```

Figure 6 — Example 3: Get the list of harvest requests (the harvest trail).

NOTE 2: The correlation between a harvest request and the resulting record(s) is advertised via the "item" link(s). The link to the resulting record(s) could also be a query if the number of resulting records is large. That would likely require a different rel (`data` perhaps).

5.2.3.5. Demonstration

The demonstration at the code sprint consisted of a script that would generate a harvest request and post it to the harvest endpoint of the CubeWerx server. The code of the script, named `harvest`, was:

```
#!/usr/bin/csh

echo '{"sources":[{"href": "'$1'","type":"SENTINEL1"}]}' > harvest.json
cwhttp method=POST url='https://pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/sentinel1cat/harvest?pretty=true' content=harvest.json
contentType=application/json extraheaders="Accept: text/xml"
```

The input to the script is an S3 reference to a Sentinel1 product.

The script generates a JSON harvest request and saves it to a temporary file named `harvest.json`. This file is then posted to the server's harvest endpoint at https://pvretano.com/cubewerx/cubeserv/default/ogcapi/catalogues/collections/sentinel1cat/harvest.

In response to the following invocation:

```
./harvest s3://sentinel-s1-l1c/GRD/2022/9/16/EW/DH/S1A_EW_GRDM_1SDH_
20220916T111355_20220916T111455_045029_056148_189B/
```

The server generated the following response to the harvest request:

```
HTTP/1.1 200 OK
Date: Thur, 17 Sep 2022 09:17:35 GMT
Server: Apache
Vary: Accept-Language,Origin
CubeWerx-Suite-Version: 9.5.4
Content-Disposition: inline; filename="HarvestResult.json"
Content-Type: application/geo+json

  {
    "id": "urn:uuid:00000c8e-d493-11ec-a1bc-43c1b3ebff8b",
    "type": "Feature",
    "created": "2022-09-15T09:17:34Z",
    "geometry": {
      "type": "Polygon",
      "coordinates": [[[ -80.738129, -0.875548 ], [ -78.135155, -0.875548 ],
                       [ -78.135155,  1.346873 ], [ -80.738129,  1.346873 ],
                       [ -80.738129, -0.875548 ], [ -80.738129, -0.875548 ]]]
    },
    "properties": {
      "type": "urn:cw:def:ebRIM-ObjectType:CubeWerx:DataProduct",
      "title": "SENTINEL-1 Product (S1A_EW_GRDM_1SDH_20220916T111355_
20220916T111455_045029_056148_189B)"
    }
  }
```

NOTE: The CubeWerx server will generate a feature collection if more that one record is created in response to a harvest request.

The following is the HTML representation of this complete record:

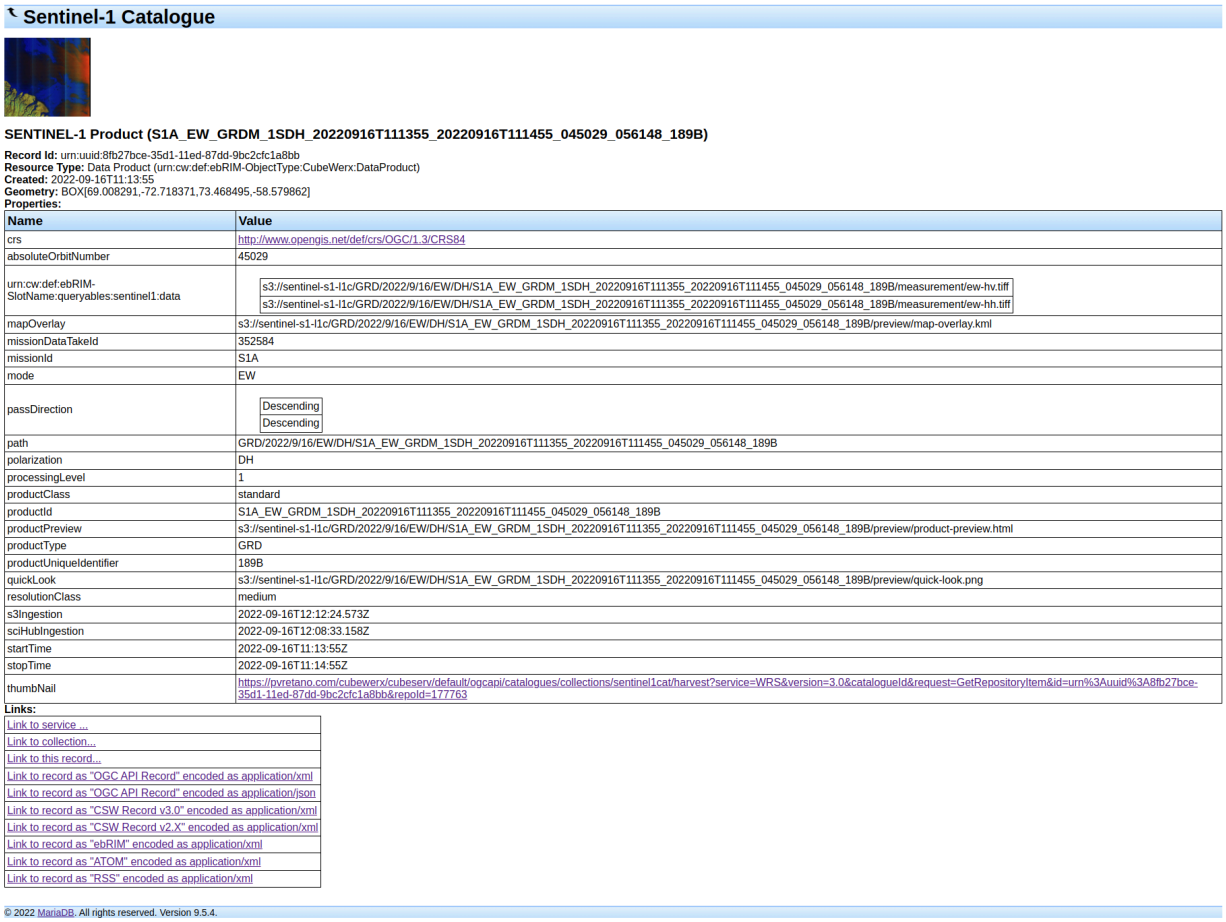

Figure 7 — Screenshot of the CubeWerx CubeSERV OGC API - Records demo

5.3. ldproxy

Before the Code Sprint, ldproxy supported the draft version of JSON-FG from January 2022.

In the Code Sprint, the implementation was updated by the participants from interactive instruments to support version 0.1, released just before the Code Sprint. In addition, support for the Polyhedron geometry type and for the `compatibility=geojson` media type parameter was added.

The updated software was deployed on two servers for use by other participants, as follows.

- The main demonstration deployment of ldproxy (demo.ldproxy.net), which includes among others the Ordnance Survey Open Zoomstack dataset.
- A deployment from Testbed 18 with CityGML building data from the City of Montreal.

A screenshot from the demo.ldproxy.net deployment is shown in the figure below.

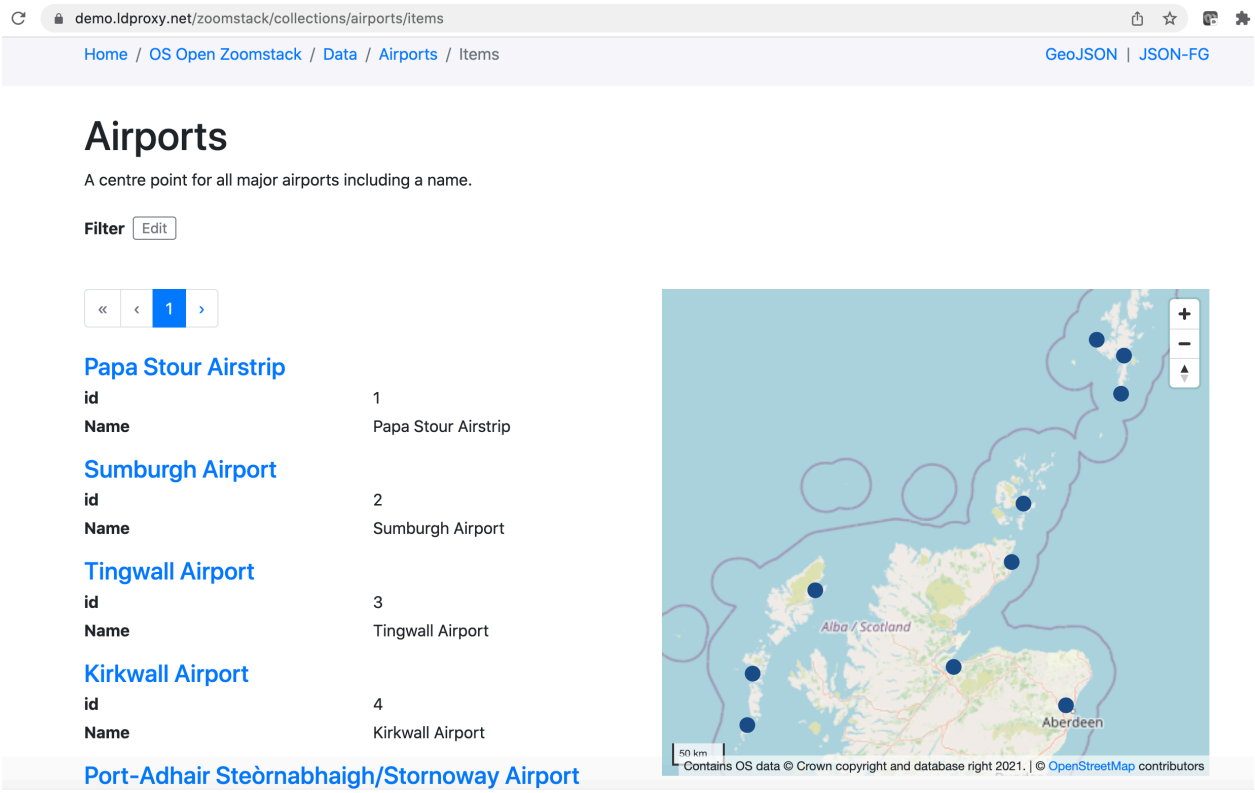

Figure 8 — Screenshot of the interactive instruments demo with ldproxy using OS Zoomstack data

The following examples illustrate the differences in responses depending on the request parameters:

Example 1 — Request three airports as GeoJSON: The request fetches three airports as GeoJSON in the default coordinate reference system WGS 84 longitude/latitude:

```
curl "https://demo.ldproxy.net/zoomstack/collections/airports/items?limit=3" -
H "accept: application/geo+json" -o demo.geo.json
```

The response consists of the standard GeoJSON response with additional members in the feature collection object specified by OGC API — Features (“numberReturned” and “timeStamp”):


```
{
    "type": "FeatureCollection",
    "numberReturned": 3,
    "timeStamp": "2022-09-28T12:50:23Z",
    "features": [
        {
            "type": "Feature",
            "id": 1,
            "geometry": { "type": "Point", "coordinates": [-1.6930015, 60.
3216821] },
            "properties": {"name": "Papa Stour Airstrip"}
        },
        {
            "type": "Feature",
            "id": 2,
```

```
            "geometry": { "type": "Point", "coordinates": [-1.2922268, 59.
8782666] },
            "properties": {"name": "Sumburgh Airport"}
        },
        {
            "type": "Feature",
            "id": 3,
            "geometry": { "type": "Point", "coordinates": [-1.2439112, 60.
191746] },
            "properties": {"name": "Tingwall Airport"}
        }
    ],
    "links": [ ... ]
}
```


Example 2 — Request three airports as JSON-FG: The request requests three airports as JSON-FG in the default coordinate reference system WGS 84 longitude/latitude:

```
curl "https://demo.ldproxy.net/zoomstack/collections/airports/items?limit=3" -
H "accept: application/vnd.ogc.fg+json" -o demo.fg.json
```

The response adds the following information to the feature collection object: The “coordRefSys” member with the identifier of the coordinate reference system (WGS 84 lon/lat) as well as the “geometryDimension” to signal to parsing clients that all features have a point geometry.

The geometry is still encoded in the “geometry” member from GeoJSON since it is in WGS84 longitude/latitude and one of the Simple Feature geometries supported by GeoJSON (points). The “place” member required by JSON-FG is added and set to `null`.

The airport features have no temporal information, the required “time” member is therefore set to `null`.


```
{
    "type": "FeatureCollection",
    "numberReturned": 3,
    "timeStamp": "2022-09-28T12:51:23Z",
    "coordRefSys": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
    "geometryDimension": 0,
    "features": [
        {
            "type": "Feature",
            "id": 1,
            "geometry": { "type": "Point", "coordinates": [-1.6930015, 60.
3216821] },
            "properties": {"name": "Papa Stour Airstrip"},
            "place": null,
            "time": null
        },
        {
            "type": "Feature",
            "id": 2,
            "geometry": { "type": "Point", "coordinates": [-1.2922268, 59.
8782666] },
            "properties": {"name": "Sumburgh Airport"},
            "place": null,
            "time": null
        },
        {
            "type": "Feature",
```

```
            "id": 3,
            "geometry": { "type": "Point", "coordinates": [-1.2439112, 60.
191746] },
            "properties": {"name": "Tingwall Airport"},
            "place": null,
            "time": null
        }
    ],
    "links": [ ... ]
}
```

Example 3 — Request three airports as JSON-FG in the British National Grid: The request requests three airports as JSON-FG in the coordinate reference system British National Grid:

```
curl "https://demo.ldproxy.net/zoomstack/collections/airports/items?limit=
3&crs=http://www.opengis.net/def/crs/EPSG/0/27700" -H "accept: application/vnd.
ogc.fg+json" -o demo.fg.json
```

The difference to the previous response is that the point geometry is now encoded in the JSON-FG "place" member since it is not in WGS 84 longitude/latitude. The "geometry" member is set to `null`.

```
{
    "type": "FeatureCollection",
    "numberReturned": 3,
    "timeStamp": "2022-09-28T12:54:35Z",
    "coordRefSys": "http://www.opengis.net/def/crs/EPSG/0/27700",
    "geometryDimension": 0,
    "features": [
        {
            "type": "Feature",
            "id": 1,
            "geometry": null,
            "properties": {"name": "Papa Stour Airstrip"},
            "place": { "type": "Point", "coordinates": [417057.93, 1159772.2]
},
            "time": null
        },
        {
            "type": "Feature",
            "id": 2,
            "geometry": null,
            "properties": {"name": "Sumburgh Airport"},
            "place": { "type": "Point", "coordinates": [439723.69, 1110559.95]
},
            "time": null
        },
        {
            "type": "Feature",
            "id": 3,
            "geometry": null,
            "properties": {"name": "Tingwall Airport"},
            "place": { "type": "Point", "coordinates": [442029.48, 1145501.02]
},
            "time": null
        }
    ],
    "links": [ ... ]
}
```

Example 4 — Request three airports as JSON-FG in the British National Grid with an additional geometry in WGS 84: The request requests three airports as JSON-FG in the coordinate reference system British National Grid, but requests to also represent the geometry in WGS 84 to support GeoJSON clients that are not JSON-FG-aware or have no support for coordinate reference systems and only support WGS 84:

```
curl "https://demo.ldproxy.net/zoomstack/collections/airports/items?limit=
3&#x26;crs=http://www.opengis.net/def/crs/EPSG/0/27700&#x22; -H &#x22;accept:
application/vnd.ogc.fg+json;compatibility=geojson" -o demo.fg.json
```

The difference to the previous response is that the point geometry is now also encoded in the GeoJSON "geometry" member.


```
{
    "type": "FeatureCollection",
    "numberReturned": 3,
    "timeStamp": "2022-09-28T12:56:06Z",
    "coordRefSys": "http://www.opengis.net/def/crs/EPSG/0/27700",
    "geometryDimension": 0,
    "features": [
        {
            "type": "Feature",
            "id": 1,
            "geometry": { "type": "Point", "coordinates": [-1.6930015, 60.
3216821] },
            "properties": {"name": "Papa Stour Airstrip"},
            "place": { "type": "Point", "coordinates": [417057.93, 1159772.2]
},
            "time": null
        },
        {
            "type": "Feature",
            "id": 2,
            "geometry": { "type": "Point", "coordinates": [-1.2922268, 59.
8782666] },
            "properties": {"name": "Sumburgh Airport"},
            "place": { "type": "Point", "coordinates": [439723.69, 1110559.95]
},
            "time": null
        },
        {
            "type": "Feature",
            "id": 3,
            "geometry": { "type": "Point", "coordinates": [-1.2439112, 60.
191746] },
            "properties": {"name": "Tingwall Airport"},
            "place": { "type": "Point", "coordinates": [442029.48, 1145501.02]
},
            "time": null
        }
    ],
    "links": [ ... ]
}
```


Example 5 — Request a 3D building as JSON-FG: The request fetches the Notre-Dame building from the CityGML LoD 2 dataset of the City of Montreal as JSON-FG:

```
curl "https://d123.ldproxy.net/montreal/collections/buildings/items/248460" -H
"accept: application/vnd.ogc.fg+json" -o demo.fg.json
```

The geometry is a Polyhedron with an outer shell consisting of many patches (shorted in the snippet below) in the JSON-FG “place” member.


```
{
    "type": "Feature",
    "featureType": "building",
    "coordRefSys": "http://www.opengis.net/def/crs/OGC/0/CRS84h",
    "id": 248460,
    "geometry": null,
    "properties": {
        "gml_id": "uuid_9ea4602f-46d1-4dae-a786-4f5f43a2b5f9",
        "creationDate": "2022-07-12",
        "categorie": "Régulier",
        "id_uev": "01091097",
        "nom": "Ville-Marie",
        "ori_bldgid": "2104973",
        "function": 6911,
        "yearOfConstruction": 1829,
        "measuredHeight": 73.196
    },
    "place": {
        "type": "Polyhedron",
        "coordinates": [
            [
                [
                    [
                        [ -73.556373, 45.504773, 84.92 ],
                        [ -73.556375, 45.504772, 83.86 ],
                        [ -73.556374, 45.504772, 83.86 ],
                        [ -73.556373, 45.504773, 84.92 ]
                    ]
                ],
                ...,
                [
                    [
                        [ -73.555965, 45.504706, 15.33 ],
                        [ -73.555977, 45.504686, 15.33 ],
                        [ -73.555989, 45.504665, 15.33 ],
                        ...,
                        [ -73.556027, 45.504676, 15.33 ],
                        [ -73.556003, 45.504715, 15.33 ],
                        [ -73.555965, 45.504706, 15.33 ]
                    ]
                ]
            ]
        ]
    },
    "time": {"instant": "2022-07-12"},
    "links": [ ... ]
}
```

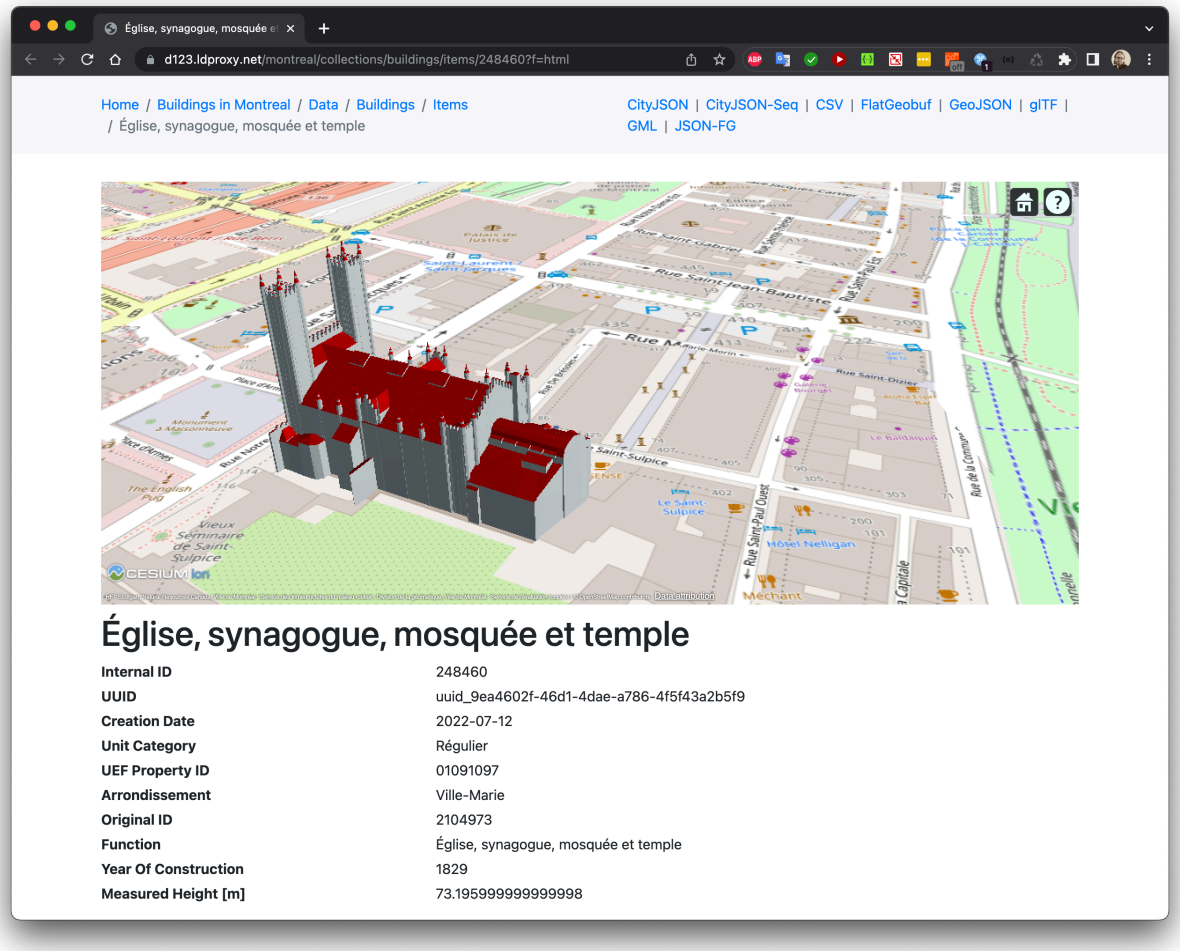

Figure 9 — The same building as HTML provided by ldproxy

All example documents have been validated using the ajv-cli tool against the JSON-FG schemas.

```
ajv validate --spec=draft2019 --validateFormats=false -s /Users/portele/Documents/GitHub/ogc-feat-geo-json/core/schemas/featurecollection.min.json -d demo.fg.json
```

The JSON-FG data from the APIs was used at least by the following clients: OSGeo Leaflet, Geonovum OGC API Testclient and the 3DGI CityJSON and JSON-FG Viewer.

5.4. GeoNetwork

The GeoNetwork team from GeoCat worked on the following topics during the Metadata Code Sprint.

5.4.1. Use of ISO 19115-3 for feature catalogues

Following discussions with participants, some would like to benefit from the possibility in ISO 19115-3 to describe the dataset's feature catalogue using the following method:

- As a citation in the dataset record (was also available in ISO 19139);
- As an embedded feature catalogue; and
- As a standalone feature catalogue.

GeoNetwork supports the previous version of ISO 19110 as a standalone record. Given that it is more relevant now to use ISO 19115-3 (which contains the latest version of ISO 19110) for standalone (or embedded) feature catalogues, the code sprint provided an opportunity to implement this capability. With that change, users can decide how to relate dataset descriptions and their corresponding feature catalogues using one of the 3 approaches above. The relevant work is in progress at https://github.com/geonetwork/core-geonetwork/pull/6545.

5.4.2. OGC API — Records implementation

During the code sprint, support for the following output formats was improved:

- GeoJSON for /items and /items/uuid; and
- DCAT for /items (not only for /items/uuid). Improved DCAT conversion.

A conformance section was also added during the code sprint. A Pull Request was created for this purpose.

Note that Elasticsearch by default returns only 10K records. A Pull Request was created for adding support for larger catalogues using track total hits.

QGIS client interaction was tested and determined to require a conformance section and GeoJSON output. The work is to be continued.

5.5. Geonovum OGC API Testclient

Geonovum worked on an OGC API client, to showcase the interaction with APIs implementing OGC API — Features and OGC API — Records. It also shows loading a JSON-FG file with a different CRSes than WGS84.

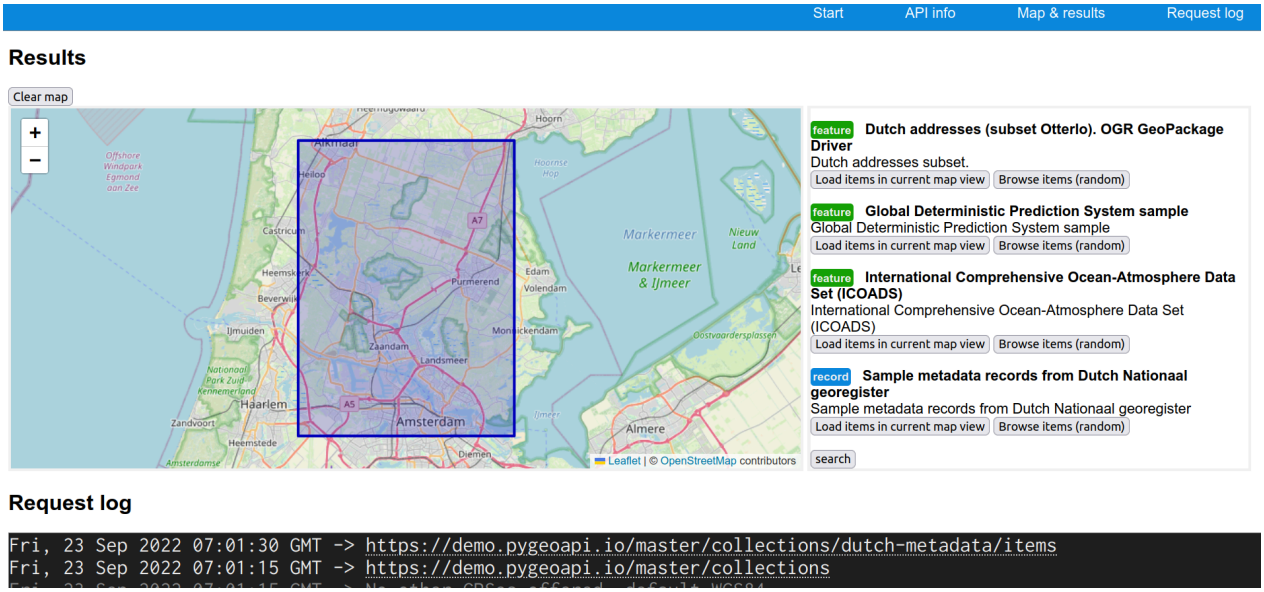

Figure 10 — Screenshot of the OGC API testclient

5.6. GeM+

GeM+ is a desktop application developed in C for managing and editing geospatial metadata. The participants from CREAF have worked on the implementation and integration into GeM+ of a client application that supports OGC API — Records. The implemented prototype of GeM+ is able to read metadata in XML format according to ISO 19115 (ISO 19139) and ISO 19115-2 by connecting to OGC API — Records. The following figures show several screenshots of the implementation in GeM+.

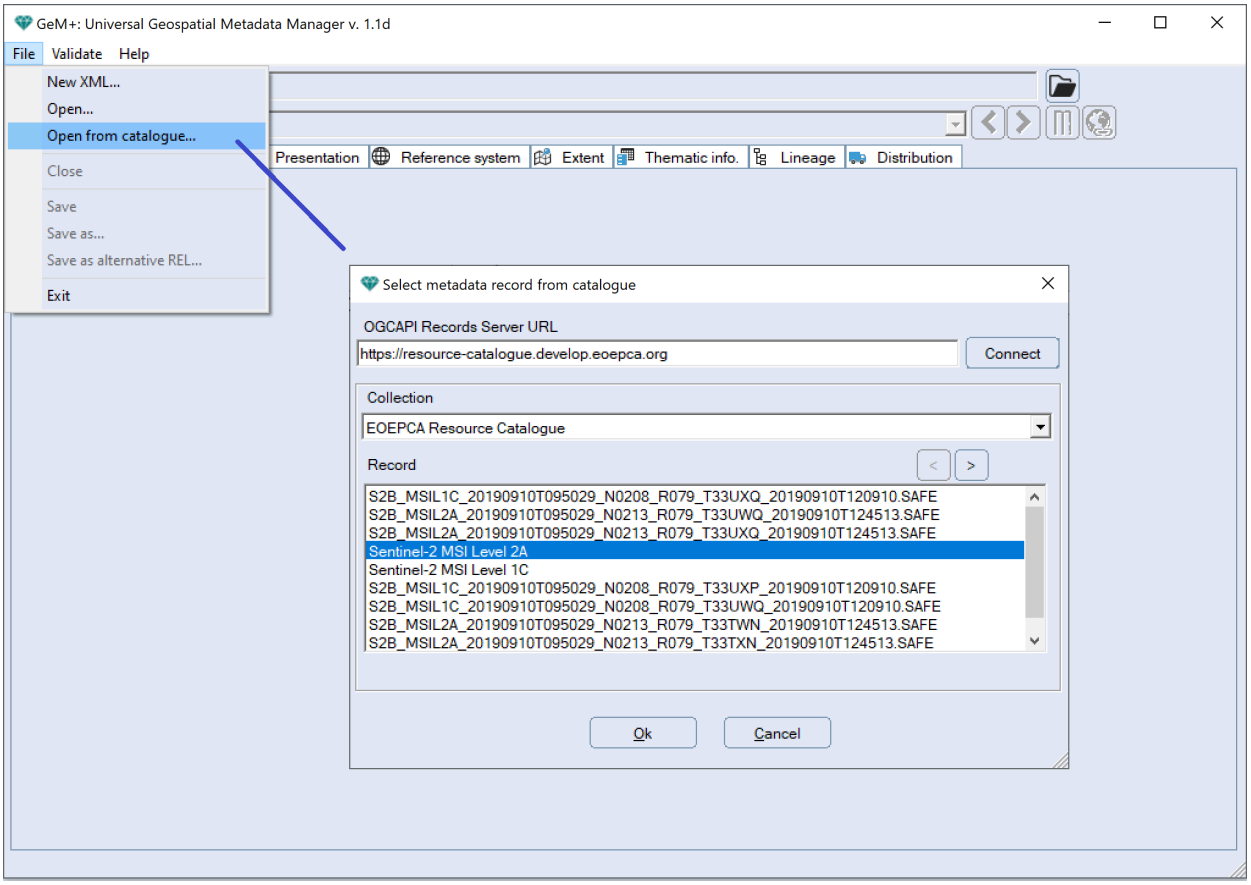

Figure 11 — Selecting a metadata record on GeM+

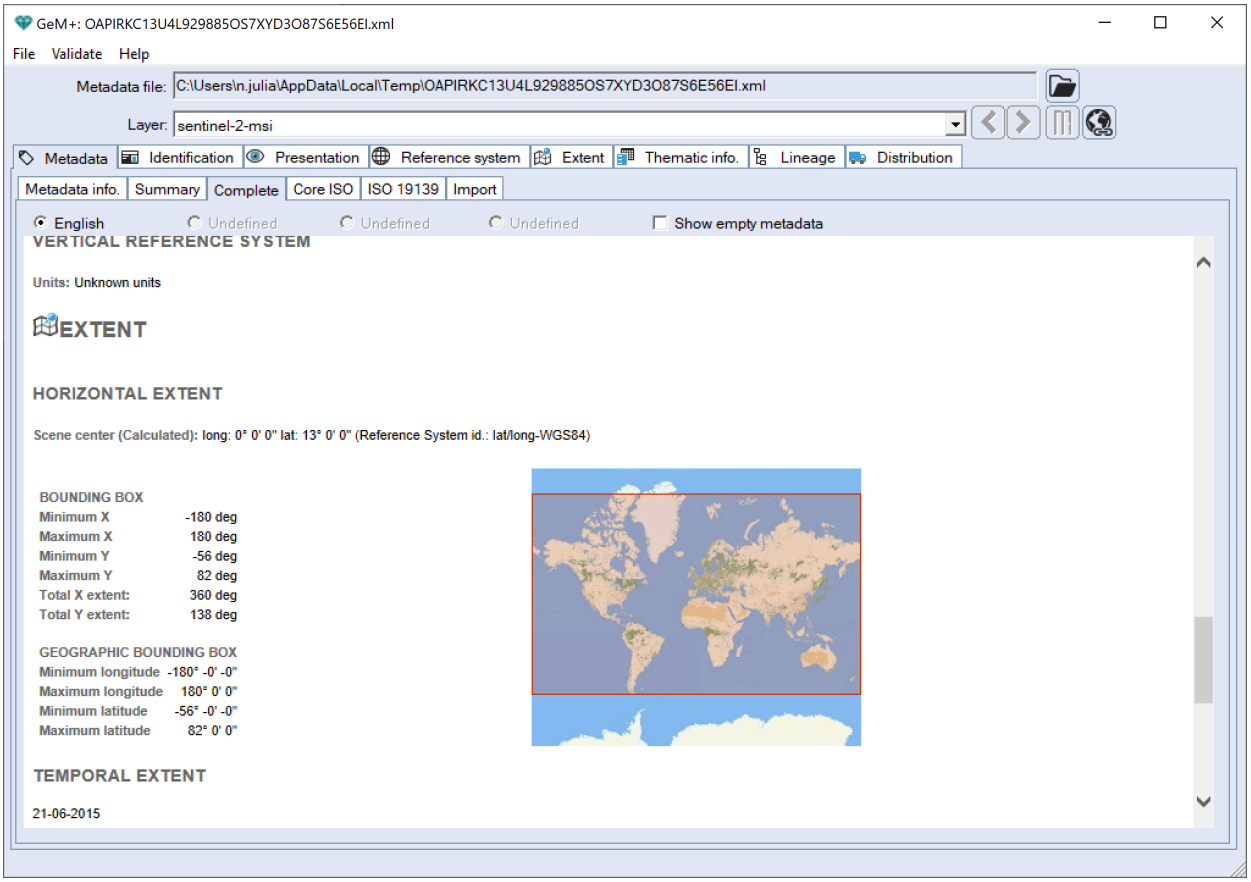

Figure 12 — Displaying a metadata record on GeM+

5.7. pygeoapi

The pygeoapi project implemented capability for metadata transactions following the OGC API — Records — Part 4: Create, Replace, Update and Delete draft specification. Support for the "Requirements Class "Create/Replace/Delete" was implemented, reviewed and approved by the pygeoapi development team. As a result, pygeoapi implementations are now able to easily define "editable" resources for metadata (OGC API — Records) as well as data (OGC API — Features) thanks to the building block approach of OGC API Standards.

The official pygeoapi documentation provides more information on how to enable the new functionality. A screenshot of the associated OpenAPI/Swagger interface from the implementation is shown in the figure below.

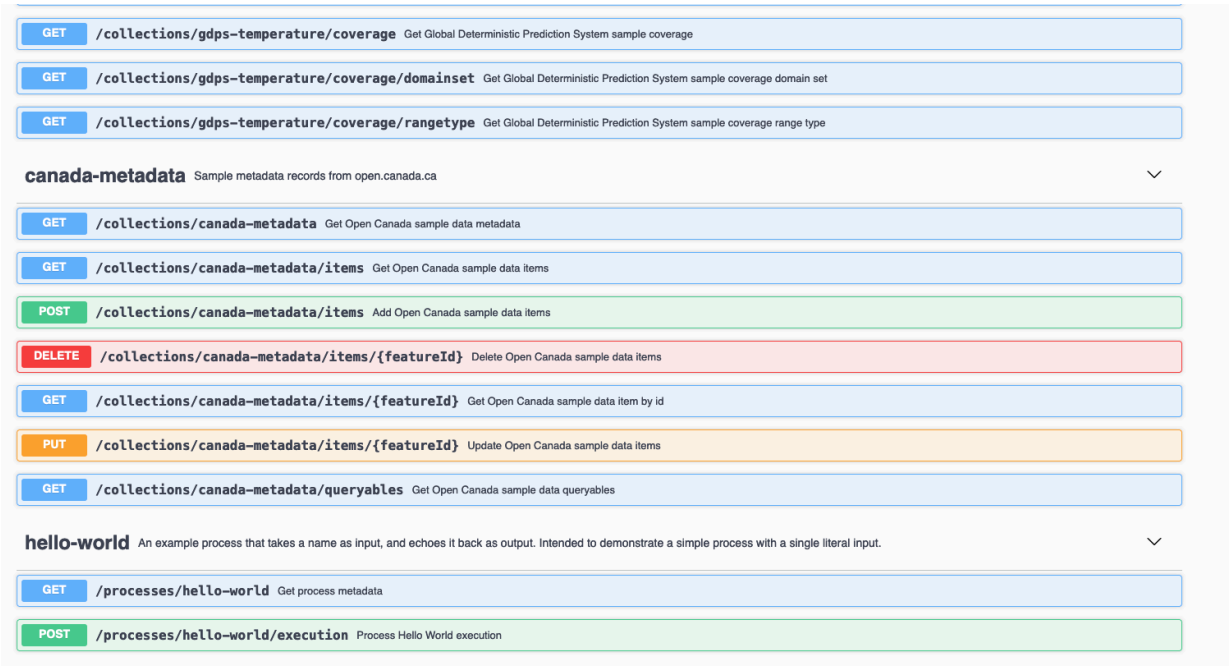

Figure 13 — Screenshot of the pygeoapi transactions demo

pygeoapi also added support for STAC-based link relations (`rel=root`) in support of interoperability with STAC tooling.

5.8. pycsw

5.8.1. Transactions support

The pycsw project implemented capability for metadata transactions following the OGC API — Records — Part 4: Create, Replace, Update and Delete draft specification. Support for the "Requirements Class "Create/Replace/Delete" was implemented, in review and expected to be included in the main codebase. The functionality leverages pycsw's existing underlying transactional support made available by transactional CSW. As a result, pycsw implementations are now able to provide OGC API based transactional support for metadata management via OGC API — Records.

A screenshot of the associated OpenAPI/Swagger interface from the implementation is shown in the figure below.

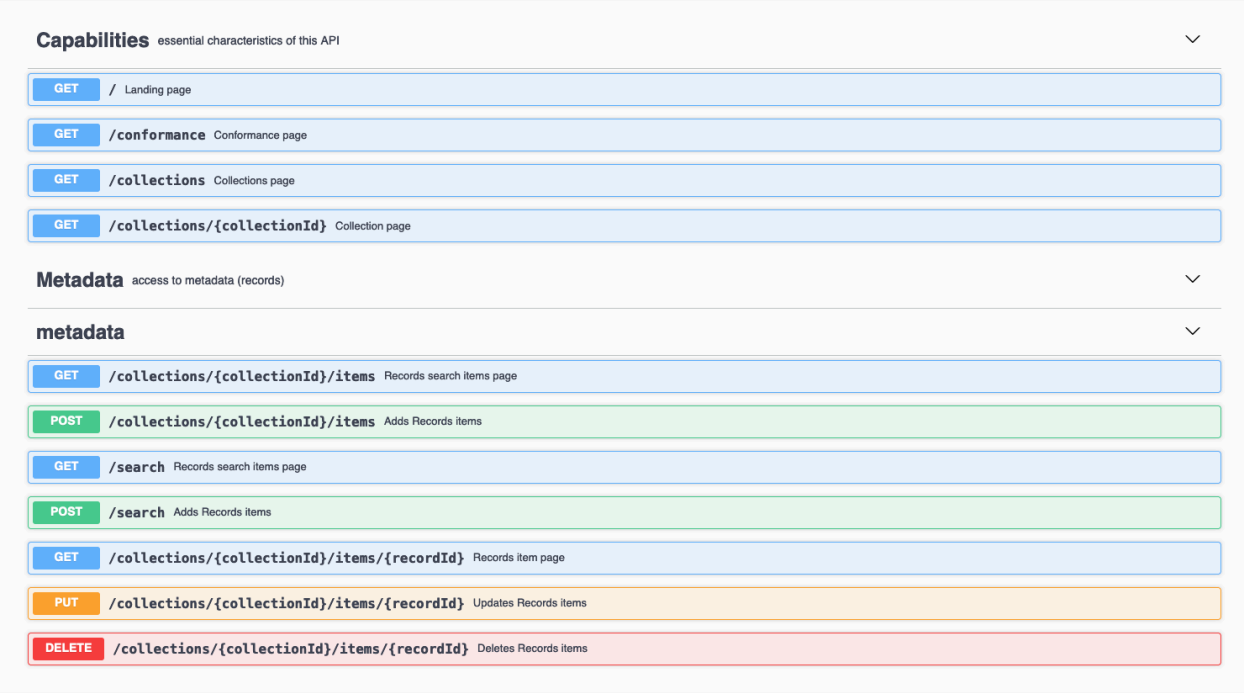

Figure 14 — Screenshot of the pycsw transactions demo

5.8.2. OGC Testbed-18 instance of pycsw

An instance of pycsw, implemented for Testbed-18, was included in the code sprint to support a demonstration of asynchronous catalogues that implement OGC API — Records.

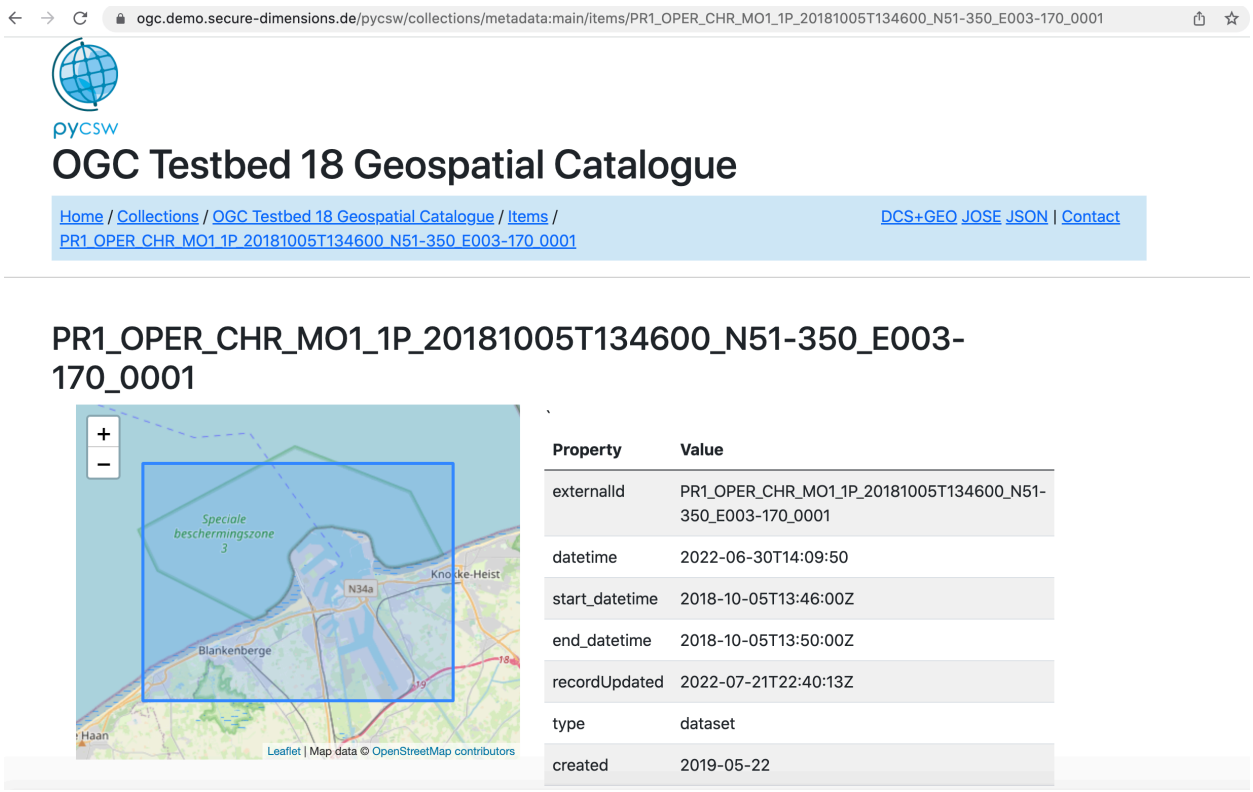

Figure 15 — Screenshot of the instance of pycsw

5.9. OWSLib

The OWSLib project implemented capability for client transactions following the OGC API — Records — Part 4: Create, Replace, Update and Delete draft specification. Support for the "Requirements Class "Create/Replace/Delete" was implemented, reviewed and approved by the OWSLib development team. As a result, Python clients can now use OWSLib for simple and Pythonic workflows to handle resource transactions for metadata (OGC API — Records) as well as data (OGC API — Features).

A screenshot of a sample Python/OWSLib workflow of the new functionality is shown in the figure below.

```
import json

from owslib.ogcapi.records import Records

record_data = 'sample-record.json'

url = 'http://localhost:8000'
collection_id = 'metadata:main'

r = Records(url)

cat = r.collection(collection_id)

with open(record_data) as fh:
    data = json.load(fh)

identifier = data['id']

r.collection_item_delete(collection_id, identifier)

# insert metadata
r.collection_item_create(collection_id, data)

# update metadata
r.collection_item_update(collection_id, identifier, data)

# delete metadata
r.collection_item_delete(collection_id, identifier)
```

Figure 16 — Screenshot of the OWSLib transactions demo

5.10. GeoPython stack

The GeoPython suite of tooling was integrated and demonstrated as part of a Mentor Stream during the code sprint in order to demonstrate end-to-end metadata lifecycle management.

- pygeometa: create / manage metadata from simple YAML configurations, exporting to numerous metadata formats
- OWSLib: publish metadata via OGC API — Features — Part 4: Create, Replace, Update and Delete to an OGC API — Records server
- pygeoapi: serve published metadata via OGC API — Records
- pycsw: serve published metadata via OGC API — Records

- QGIS: use QGIS' MetaSearch capability to discover resources via OGC API — Records

A screenshot from the mentor session is shown in the figure below.

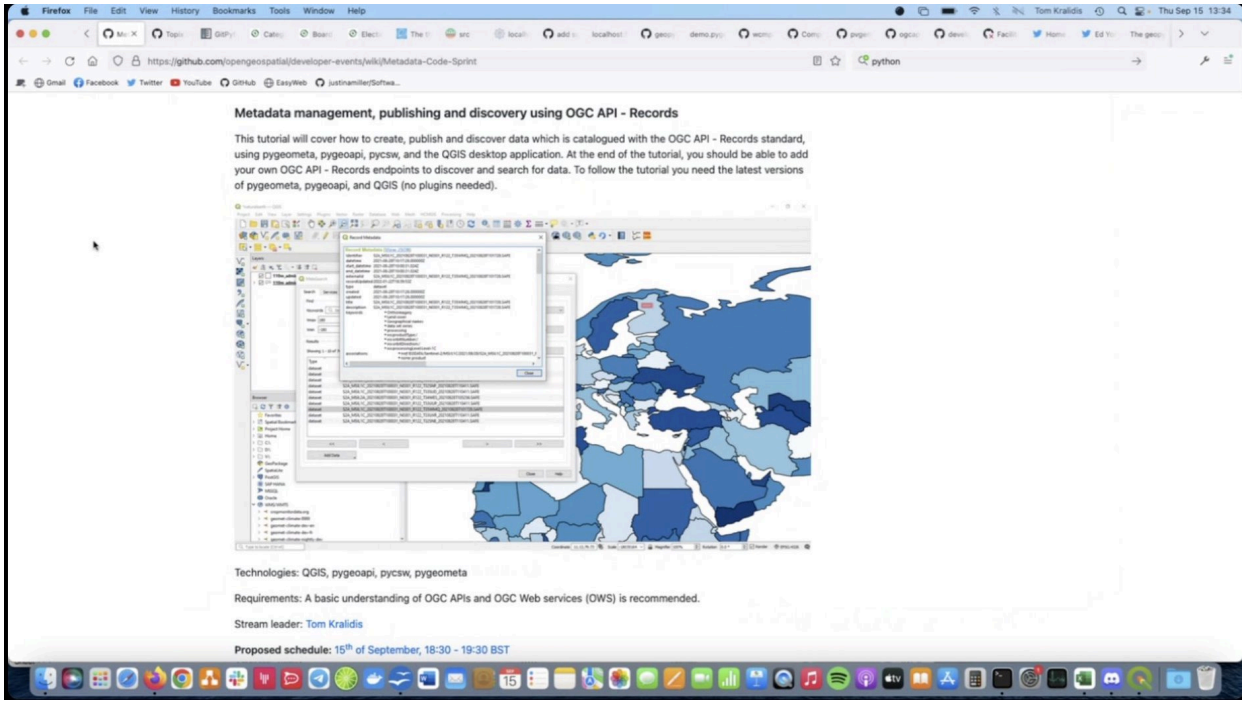

Figure 17 — Screenshot of metadata lifecycle management mentor stream using pygeometa, OWSLib, pygeoapi/pycsw, and QGIS

5.11. ISO 19115 activity by OpenWork

5.11.1. Background

OpenWork participated in the code sprint, supported by both the OGC and the Australia and New Zealand Intergovernmental Committee on Survey and Mapping (ICSM). Byron Cochrane, a metadata expert from OpenWork, participated on behalf of OpenWork with the primary role in the code sprint of sharing technical expertise on ISO 19115 metadata. This expertise included participation in standard development, implementation, development of tools and guidance. A significant part of this expertise has been gained through involvement in the ICSM Metadata Working Group (MDWG) and chairing of the OGC Metadata and Catalogue Domain Working Group, as well as participation in the OGC API — Records Standards Working Group.

One of the challenges of assisting and advising organizations in creating, managing and using spatial metadata is that due to the apparent proliferation of solutions, it has become more challenging to advise on which metadata solutions are best suited to a particular organization's needs. This is partly because the role of metadata and catalogues is multifaceted and varied across domains. Currently, unfortunately, some of the communities providing the tools or

solutions do not offer such guidance. A framework by which such guidance could be created and shared was a primary goal of OpenWork's participation.

Other sprint goals included to:

- Examine the possibility of creating a JSON encoding of ISO 19115-1; and
- Create Crosswalks between ISO 19115-1, STAC and OGC API Records.

5.11.2. Day 1

The focus of Day One of the code sprint was for participants to better understand the primary components: OGC API — Records, STAC, ISO 19115, and JSON-FG, as well as their histories, capabilities, use and future development plans. OpenWork led an hour-long presentation and discussion on ISO 19115 and related standards.

5.11.2.1. Presentation of ISO 19115

OpenWork briefed the sprint participants, highlighting issues around the lack of understanding of the differences between the standard names and versions of ISO 19115. In Australia and New Zealand, ICSM strongly endorsed the ISO 19115-1 metadata standard and deprecated the previous ISO 19115 profiles. This support includes the latest version, ISO 19115-1:2014/Amd2:2020, which provides improved support for capturing information related to coordinate drift due to tectonic movement — a significant issue in Australia where coordinate values can change over one and a half meters in 20 years. However, it is clear that many professionals confuse the versions and do not understand why changes exist. For example, a common confusion is referencing ISO 19115:2003 as ISO 19115-1 (initially likely to contrast with ISO 19115-2:2009 Extensions for imagery and gridded data). Yet ISO 19115-1 was not released until 2014. As a result, today, many software vendors claim support for ISO 19115-1 when they only support the older ISO 19115:2003 standard. Byron Cochrane explained that, for Europe, this issue is aggravated by the fact that the INSPIRE framework of the European Union (EU) has yet to provide a mechanism to support ISO 19115-1 even though work on this standard was well underway when the EU launched INSPIRE. The INSPIRE community must address this situation as it prohibits many in Europe from using the improved standards and limits software developers' willingness to include support for enhanced standards worldwide.

Two significant discussions resulted from this presentation. The first involved the value of namespaces in encoded metadata. The namespace issue is closely related to the desire to create a canonical JSON encoding of ISO 19115-1. One position was that many JSON developers dislike using namespaces and tend to remove them if they are present. A contrasting view posited that namespaces are an elegant approach for reducing ambiguity in data and, when not used, are often eventually replaced by less friendly solutions. A question raised and unanswered was whether a canonical encoding of ISO 19115-1 would be possible.

This first discussion led to the second. The question was, "What would the best approach to a JSON encoding of ISO 19115-1 be?" Would it be best to create a fully compliant canonical version like ISO 19115-3 encoded in JSON? Or would it be better to take a more piecemeal approach and extend OGC API — Records where needed until it could provide a fully compliant

record? The challenges of directly aligning the GeoJSON format used by OGC API — Records with ISO 19115-1 and the desire to improve crosswalks for at least some elements during the code sprint led OpenWork to take the latter approach. This proved to be the right approach for this sprint's shorter-term purposes and outcomes.

5.11.2.2. Crosswalks

For the remainder of Day 1, taking the advice and support of participants from MariaDB CubeWerx and the Meteorological Service of Canada (MSC), OpenWork undertook an effort to improve the ability to create crosswalks for keywords in ISO 19115-1 to OGC API — Records. OGC API — Records offers two possible locations for Keywords — Keywords and Themes/Concepts. Themes/Concepts were chosen as the natural fit. OGC API — Records follows the pattern of DCAT in this. The "Keywords" class is to hold free text tags. The Theme/Concepts can also contain free text but primarily support keyword concepts sourced from controlled vocabularies. (This could potentially confuse developers, thus one of the classes might need to be considered for removal.)

By the end of the evening, OpenWork had altered the YAML schema to support the ingestion of ISO 19115-1 keyword elements with controlled vocabularies. OpenWork also proposed changing the names of the relevant classes for clarity suggesting "keywords" be "tags" and "themes" be "keywords". In addition, to provide a method to retrieve the original entire ISO 19115-1 record, OpenWork proposed a new element, "source", to link a complete ISO 19115-3 record, should it exist. This latter recommendation was rejected in favor of guidance on using the existing "link" class to provide this functionality.

5.11.3. Day 2

On Day 2, OpenWork focused on creating a Record document conforming to the OGC API — Records candidate standard to test the guidance that OpenWork had developed the previous evening. Several changes ensued, mainly relating to the ability to provide resolvable URIs to both the concepts and their controlled vocabularies. There is a need for more work and testing.

Discussions clarified the roles that traditional catalogues, OGC API — Records and STAC serve. These can be summarized as follows.

- STAC is most appropriate for fine-level metadata — particularly cataloguing scenes in a collection of images to allow their discovery.
- Part 1 (the Core part) of OGC API — Records focuses on the general discovery of spatial resources of any type. Because of this, it may not be useful or appropriate to capture in the Core part of OGC API — Records any ISO 19115-1 metadata that exists for purposes other than discovery. However, future extensions of OGC API — Records may support purposes other than discovery.
- ISO 19115-1 exists to document and manage a resource fully. Provenance, structural, informative, administrative and other metadata elements exist in this standard catalogue. This resource is most useful after discovery.

5.11.4. Day 3

The focus of OpenWork on Day 3 was to start providing a framework to guide the use of OGC API — Records. This followed the early discussion regarding how much structure should be encoded in the standard. Creating a rigid standard was not seen as desirable. However, a guidance document that could provide authoritative direction would be acceptable. The immediate aim was to support crosswalks from ISO 19115 through this approach.

On Day 3, OpenWork focused on the keyword (themes) class for which Byron Cochrane had created crosswalks earlier in the sprint. Inclusion of additional classes and crosswalked elements could be added over time. A pattern language approach, applied in OpenWork's work with ICSM, was used as a template. Such documentation, while useful, requires a great deal of effort to produce. But it allows the schema to remain simple. However, automated validation is difficult to provide through this approach.

5.12. 3DGI CityJSON and JSON-FG Viewer

The viewer was extended to support the loading of JSON-FG files, the querying of the feature data, and the display of the feature data. The viewer was implemented to support polyhedrons, as well as multipolygons. Figure 18 shows a screenshot of the 3DGI CityJSON and JSON-FG Viewer.

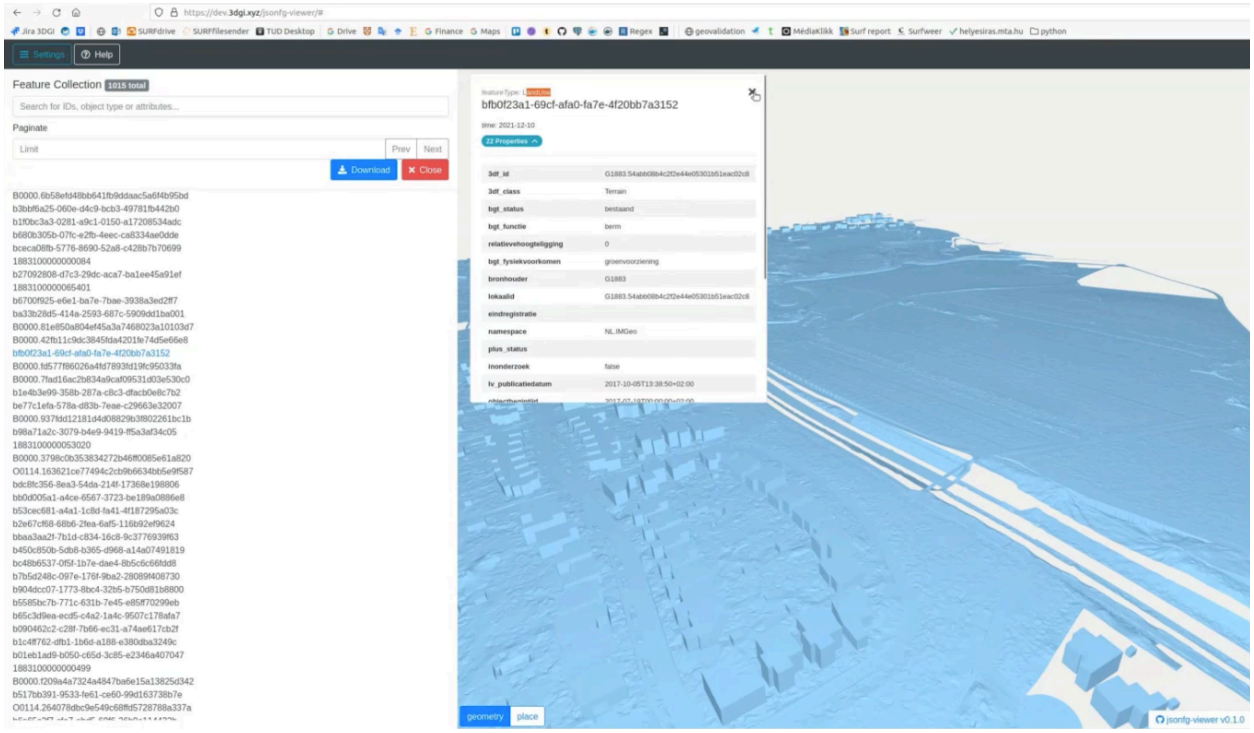

Figure 18 — Screenshot of the 3DGI CityJSON and JSON-FG Viewer

The viewer has been deployed at https://dev.3dgi.xyz/jsonfg-viewer.

5.13. Secure and Asynchronous Catalogue

During the Code Sprint, a demo was implemented that introduces the use of WebPush to a Web Browser / device to notify the user when a Catalogue resource changes. To make the demo work, a simple client application was implemented that allows the use of Web Push notifications to the Google Chrome and Firefox Web Browser. The demo does not currently work on devices running MacOS with Safari and mobile devices operating iOS 15. To make the demo work in a Chrome Browser on a device running MacOS, WebPush settings must be enabled in “notifications and alerts.”

The focus was to demonstrate the ability to use WebPush with OGC API — Records. The actual demo does not demonstrate the use of the Prefer header; instead, the demo focuses on creating a subscription with WebPush notification.

From the Secure Dimensions index page of the OGC Testbed 18 contributions, you can access all relevant services.

A subscription can be created via the OpenAPI definition documentation page by pasting a JSON object representing the Web Push details and providing a resources-uri. The resources-uri is the OGC Records API URL that is to be observed for changes.

(1) Use Google Chrome or Firefox browser and go to https://ogc.demo.secure-dimensions.de/sms/api.

(2) Click the button “ENABLE PUSH MESSAGING” which will display the web push subscription details such as


```
{
    "endpoint": "https://updates.push.services.mozilla.com/wpush/v2/gAAAAABjJD6BS4NUg76G5ozRH5kwnR5sl3cT4RwCWLwtDjK8ebnwnm53lU_M3ppWTPE_3_JyQdDd3hZnaHZ90M_vZlrFXzu4W9GN2Xakxplk943b1R4J8vRATmL4Ny0chjrZfBmLuo8x9IzTHQf-7BxuGh43stVak926qFfkHGd90R8K8k8sJjo",
    "expirationTime": null,
    "keys": {
        "auth": "HmhIs6yKqb-EgVRkYstsvQ",
        "p256dh": "BNmU2tQiJc5R3vM2hPHz9vPvxxkakPtp93F2I90H-JQPjpjWbaVhKeMl-h8ERQUlcjCaswvzx1Qq0B0Lei4jsNk"
    }
}
```


(3) Obtain an access token from the OGC Testbed Token APP.

(4) Copy the access token into the Swagger form (click “Authorize” and paste the access token).

(5) Go to the POST operation to create a subscription.

(6) Copy the following JSON object (including the web push details from above). Make sure you replace this information with the information displayed in your Web Browser.


```
{
```

```
    "name": "mySubscription",
    "delivery": "webpush://",
    "schedule": "* */1 * * *",
    "resources-uri": "https://ogc.demo.secure-dimensions.de/pycsw/collections/
metadata:main/items/e17fe655-987c-4c5f-bbae-b10dcd4fccc3?f=json",
    "expires": 1759104237,
    "sec-opts": {
        "push-subscription": {
            "endpoint": "https://updates.push.services.mozilla.com/wpush/
v2/gAAAAABjJD6BS4NUg76G5ozRH5kwnR5sl3cT4RwCWLwtDjK8ebnwnm53lU_M3ppWTPE_3_
JyQdDd3hZnaHZ90M_vZlrFXzu4W9GN2Xakxplk943b1R4J8vRATmL4Ny0chjrZfBmLuo8x9IzTHQf-7
BxuGh43stVak926qFfkHGd90R8K8k8sJjo",
            "expirationTime": null,
            "keys": {
                "auth": "HmhIs6yKqb-EgVRkYstsvQ",
                "p256dh": "BNmU2tQiJc5R3vM2hPHz9vPvxxkakPtp93F2I90H-
JQPjpjWbaVhKeMl-h8ERQUlcjCaswvzx1Qq0B0Lei4jsNk"
            }
        }
    }
}
```


The created subscription will run every 1 minute and compare the result from the resources-uri. You should receive one notification after 1 minute unless the record changes.

(7) To start the subscription, you need to do a state change via the PATCH operation. You can paste the following JSON to start the subscription. For all states of the subscription, please observe this diagram.


```
{
   "state": "start"
}
```


After one minute, you should see the notification in your browser / on your device. Clicking on the notification will open a new Web Browser window opening the resources-uri from the subscription.

5.13.1. Follow-Up

The results of the Code Sprint will be contributed to Testbed 18. In particular, more details will become available in the Testbed 18 Engineering Report “Secure and Asynchronous Catalogue”. Also, the demo of using the *Prefer* header with OGC API Records to control the asynchronous follow-up communication, will be available with the Testbed 18 results.

5.14. dataset-tagger

During the code sprint, contributors to the dataset-tagger project added support for the handling of metadata served by implementations of the OGC API Records. Figure 19 shows a screenshot of the application after importing an ISO 19139 metadata document from the *Nationaal Georegister* of the Netherlands.

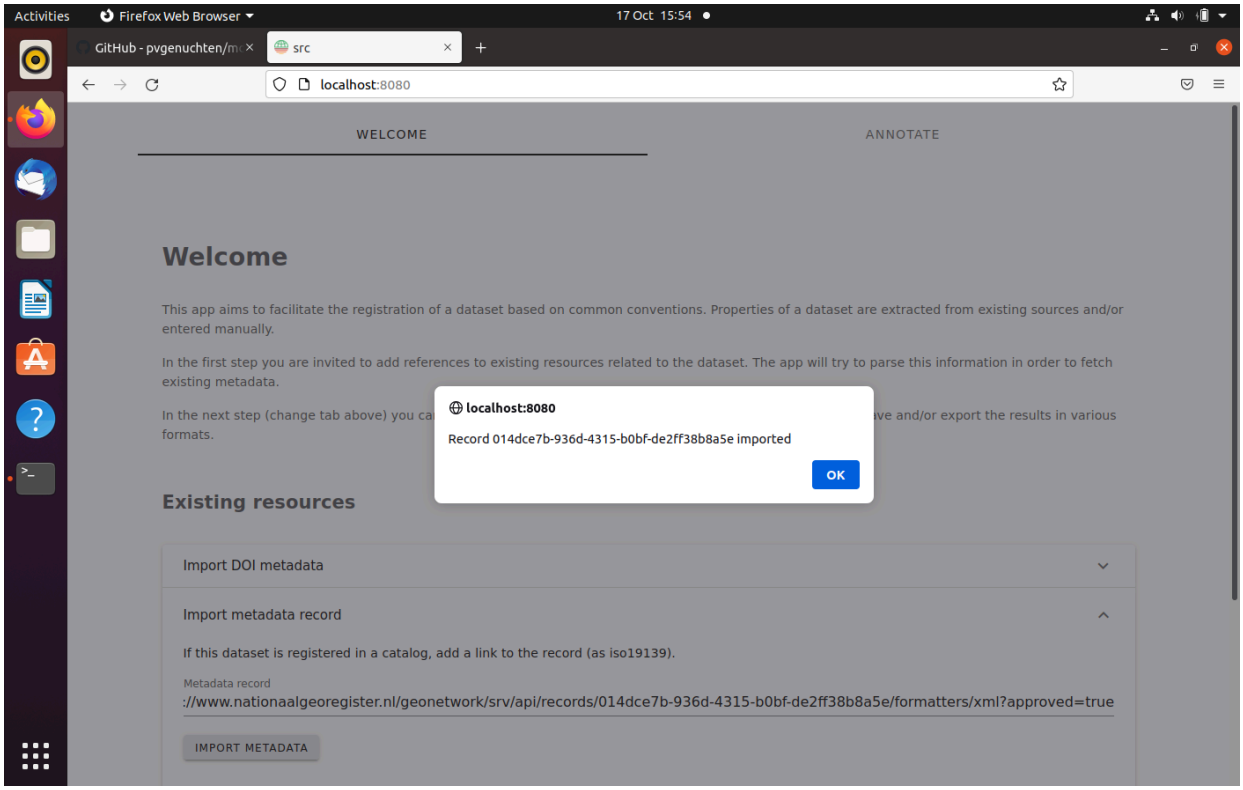

Figure 19 — Screenshot of the dataset tagger

5.15. University of Manchester Natural Language Processing for Spatial Analysis

The team from the University of Manchester implemented a Natural Language Processing (NLP) toolkit for conducting spatial analysis. The toolkit was configured to ingest feature data represented as a GeoJSON FeatureCollection. The team experimented with the use of OGC API — Features for serving feature data to an implementation of the Leaflet client application. A screenshot showing the toolkit is presented in Figure 20.

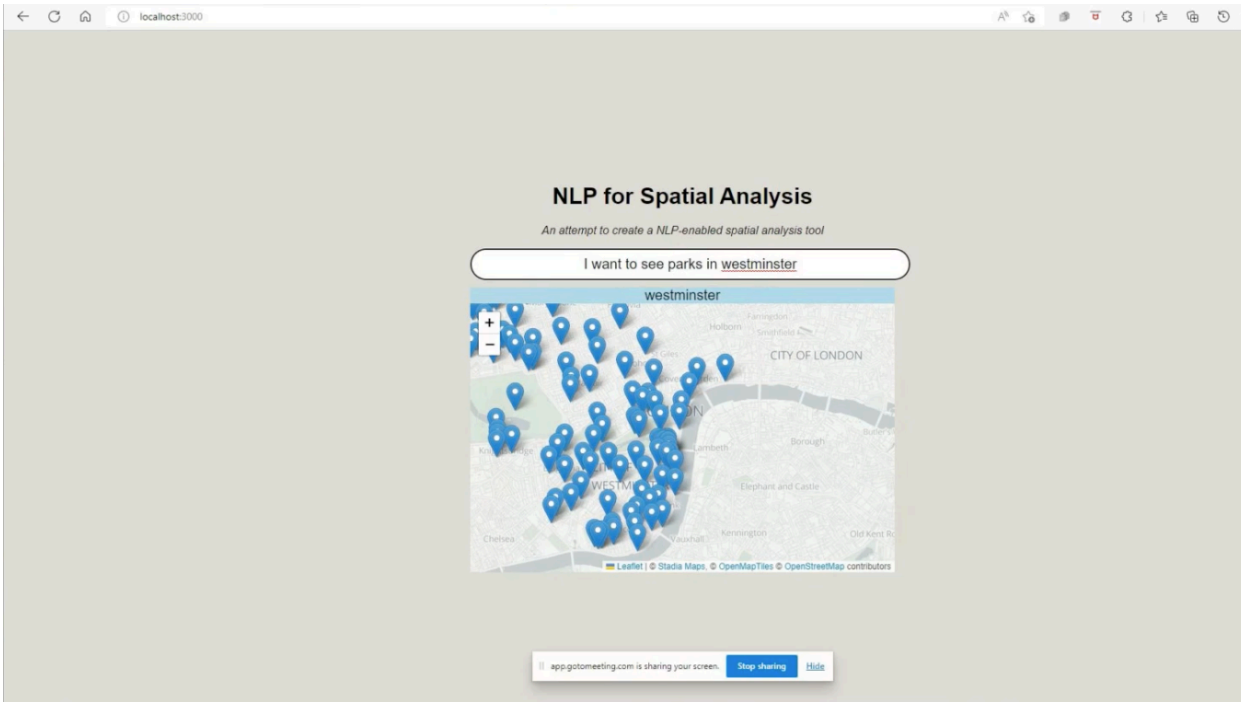

Figure 20 — Screenshot of the Natural Language Processing for Spatial Analysis toolkit prototype by the University of Manchester

5.16. STAC Browser

During the code sprint, an implementation of the STAC Browser was configured to enable access to implementations of OGC API — Features and OGC API — Records. The STAC Browser is a web application for browsing static STAC catalogs and STAC APIs. Figure 21 shows a screenshot of the STAC Browser accessing an implementation of OGC API — Features.

Figure 21 — Screenshot of the STAC Browser accessing an implementation of OGC API - Features

Figure 22 shows a screenshot of the STAC Browser accessing an implementation of OGC API — Records.

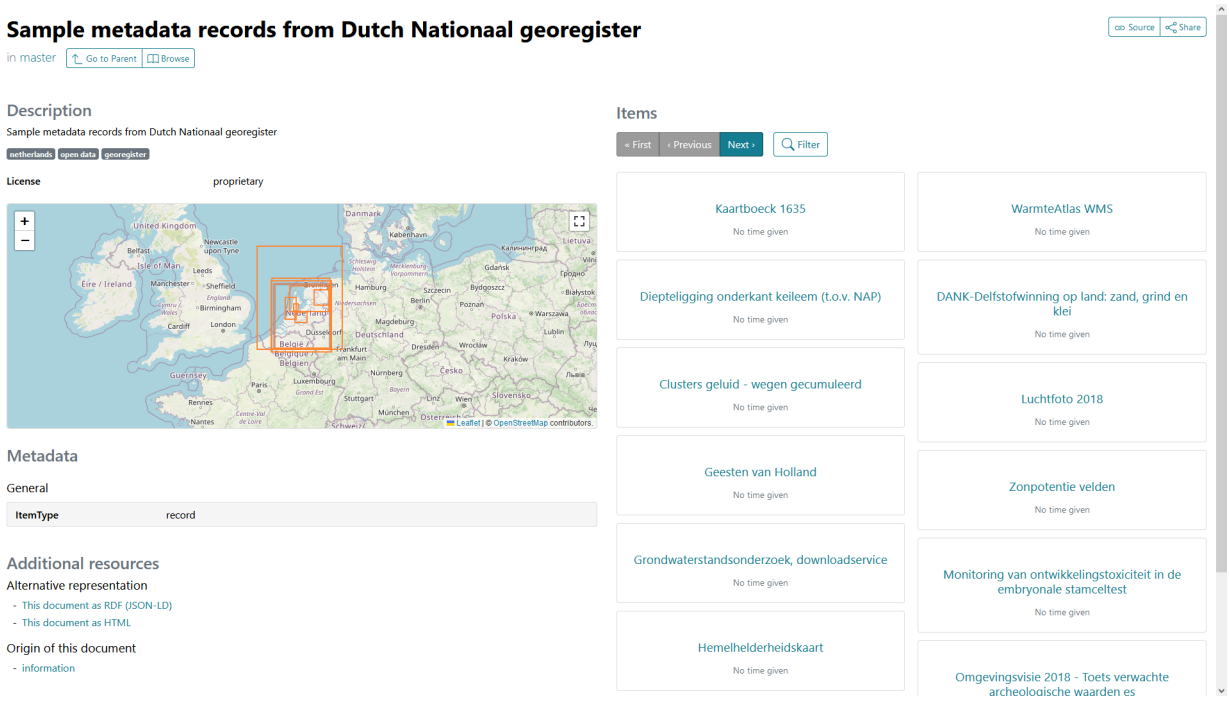

Figure 22 — Screenshot of the STAC Browser accessing an implementation of OGC API - Records

6

DISCUSSION

6 DISCUSSION

6.1. Harmonization between STAC and OGC API Records

Although both STAC and OGC API — Records use GeoJSON for encoding metadata, alignment is necessary in order to improve metadata exchange. STAC has an Asset construct and OGC API — Records uses a Link construct. If OGC API — Records adopts the use of Assets, then there is a need to be clear about the difference between Assets and Links.

There is a perception that STAC is focused on Earth Observation (EO). The Record metadata model which is specified in OGC API — Records is supposed to cover more than just EO. The idea with the Records metadata model is that it offers a small set of generic properties that can be used to describe anything. This makes it possible for communities of interest to extend the model to support their particular use case. The way that one uses a Record depends on how the person wants to make the resource discoverable. This impacts how the record is created and how it is linked with other Records.

There is a conflict between some elements where they may have the same name but different meaning (e.g., created and updated dates in STAC and OGC API — Records). The Records metadata model targets a resource, whereas STAC targets a distribution. This creates a challenge for Records because it would be impossible to give different dates of update for different distributions of the same resource. To align OGC API — Records with STAC there would be a need to change 'record-created' and 'record-updated' fields to simply 'created' and 'updated'. Other potential opportunities for alignment include the use of a 'roles' element in the links section.

It was also determined that the STAC 'root' link relation types need to be further clarified as OGC link relation types or Compact URIs (CURIEs).

Every Item has a link to one collection. You could create a hierarchy using a collection. One level of collection and then the items are records. In this context, STAC and OGC API — Records are already aligned.

The lessons identified regarding harmonization can be summarized as follows.

- Modify the create and updated fields by removing them from the Record class and add them to the links. This would improve alignment with STAC.
- In some cases, it may be necessary to use either STAC or OGC API — Records, depending on the needs of the community of interest.
- Clarify definition and intent of STAC 'root' link relation types in relation to OGC link relation types or Compact URIs (CURIEs).

6.2. Harvesting

It is important to be clear about what harvesting means. It is the complementary operation to Transactions. In Transactions you push a record to the catalogue, that requires the client to take the source and push that to the catalogue. For harvesting, you tell the catalogue "here is the resource" and the catalogue extracts the metadata and takes care of the rest. It is always possible that the catalogue will go to the resource and not know what to do with it. In the CSW standard there was support for harvesting. With harvesting the onus is on the catalogue to determine how to transform the resource and create the record for that resource.

There is also the question of: if the user points the harvester to a resource through the client, should the user leave the transformation of the resource to the harvester? The simplest approach is to leave it up to the harvester once the client has directed the harvester towards a resource. In the CSW-ebRIM model there was guidance on how to harvest a GetCapabilities response. As a minimum they would look for all of the fields that are in the record and try to populate those. More guidance would be needed for OGC API Standards, for example how deep to go in navigating the link graph.

An example workflow could be: The client application submits a harvest request, with a prefer header, then the request would be executed synchronously or asynchronously. The server would notify the client application that it has initiated a job, meaning that the client application can monitor the job and retrieve the results when the job is complete. To achieve such a workflow, there is a need to define an API for harvesting resources or collections of resources. Providing guidance on how to crawl an OGC API resource tree and to harvest the resources it offers.

There are some aspects of harvesting that are shared with transactions. For example, the ability to create a record is an aspect of both harvesting and transactions. There are some aspects, for example 'deletion', however of transactions that are less likely to be used in harvesting. Currently, OGC API — Features — Part 4: Create, Replace, Update and Delete defines two requirements classes ("Create/Replace/Delete", and "Update"). Given that the intention is for OGC API – Records to leverage the capabilities specified by OGC API — Features — Part 4, it would be valuable to split the "Create/Replace/Delete" requirements class into 3 separate classes, to allow for finer granularity for resource management. This would make it possible for a harvester to only implement the ability to 'create' a record, without needing to implement the ability to replace nor delete a record.

The lessons identified regarding harvesting can be summarized as follows.

- It is important to be clear about what harvesting means.
- Keeping the metadata close to the data is more efficient than copying the metadata to a separate server. However, there is a need to be clear about what is meant by "close to the data."
- Ideally harvesting would be of selected bits of metadata instead of the complete metadata record.

- There are different types of harvesting, in some cases there may be some processing needed. One type of approach means harvesting the discovery metadata.
- In some cases, augmented metadata may need to be pushed back to the source.
- It would be valuable to split the "Create/Replace/Delete" requirements class of OGC API – Features – Part 4 into 3 separate classes, to allow for finer granularity for resource management.

6.3. ISO 19115 metadata and OGC API Records

There was a view put forward that the metadata should be managed with the data. It should not be stored away from the dataset it describes. If one wants to harvest ISO 19115 metadata, then they should be able to retrieve it from the API. Currently there is little guidance of how to harvest or link ISO 19115 metadata from the API definition documents or the API landing pages. There was an alternative view that often people want to catalogue things that they do not have access to, for example, for future reference. In some cases, the resource being catalogue might not even be online.

One could initiate a harvest, the catalogue could then go to the product directory and then find the associated metadata, and then harvest information from that into a record, including a link to that associated metadata. This approach means that the metadata does not actually need to be stored away from the dataset it describes. The enduring question is whether or not such a workflow requires an ISO 19115 metadata profile or some custom guidance. There is also a question of how much needs to be mapped between OGC API – Records and ISO 19115.

Given the focus of OGC API – Records – Part 1 on discovery, fields that are related to 'search' are worth initial focus. That is part of the reason that the OpenWork activity in the code sprint focused initially on the Keywords class. The Keyword Type is meant to represent a category of Keywords.

The lessons identified regarding ISO 19115 metadata and OGC API — Records can be summarized as follows.

- Expressing ISO 19115 metadata in OGC API Records should focus on discovery elements.
- Initial prototyping has been focused on Keywords to Themes.
- What is needed is a profile that enables us to work with ISO 19115.
- Content Negotiation by Profile could be useful.
- The incremental approach would be useful.
- It may be necessary to also design a JSON profile of ISO 19139 as well.
- There are various considerations relating to alignment with ISO 19115 e.g., alignment with DCAT.

There is a need to balance how deeply ISO metadata is represented in JSON. Therefore, further experimentation will be needed to arrive at a JSON encoding of ISO 19115-1 metadata.

6.4. JSON-FG

The feedback from implementations of the current draft of JSON-FG were positive, but several topics were identified where the specification should be clearer or where the language should be improved. These were recorded as issues in the JSON-FG GitHub repository with the label "2022-09 Sprint".

6.4.1. Use of Geometry Collection is underspecified

The specification needs to be clearer if and how a GeoJSON `GeometryCollection` geometry can be used in the JSON-FG "place" member.

- In version 0.1 a `GeometryCollection` is not one of the valid geometry objects in "place". See the place.json schema and requirement `/req/core/geom-valid`.
- However, since a `GeometryCollection` can be used as a value in the GeoJSON "geometry" member, it could be argued that such geometry objects should also be allowed in "place", if another coordinate reference system is used.
- If `GeometryCollection` would be allowed in "place", the current scoping rules for the "coordRefSys" member would allow that multiple coordinate reference systems could be used in different geometry objects in the collection. This should not be allowed.

6.4.2. Clarify `CustomGeometry`

The `CustomGeometry` object in the JSON schema is an extension point for additional geometry objects specified by a community or by a future version of JSON-FG. The object validates JSON files with unknown geometry types in the "place" member, these files are considered valid JSON-FG. This implements the following statement in the specification:

> JSON-FG readers should be prepared to parse values of "place" that go beyond the schema that is implemented by the reader. Unknown members should be ignored and geometries that include an unknown geometry type should be mapped to null.

However, the `CustomGeometry` object is not discussed in the specification and just used in the JSON schema. The role of the object should be clarified.

6.4.3. Always include “geometry” when “place” is not null?

Currently the “geometry” member may be `null`, if the primary geometry of a feature is in the “place” member. According to the GeoJSON specification, such features are considered “unlocated,” but the feature obviously has a location as expressed in the “place” member.

6.4.4. Additional JSON-FG metadata to simplify parsing

A proposal has been raised to add information to every JSON-FG document so that parsers can easily determine that the document is a JSON-FG document and the JSON-FG version used to encode the document. This could be addressed in multiple ways, e.g., by `"jsonfg_version": "0.1.0"` or by using the “conformsTo” member used in the OGC API standards and the URIs of the supported conformance classes.

6.4.5. Use of JSON-FG in OGC API Records and STAC

At the time of the Code Sprint, OGC API Records included a tentative requirements class with JSON-FG as an encoding of a record.

In addition, there is obvious overlap between new members specified by JSON-FG (e.g., “time”) and content in a STAC item. A general difference is that JSON-FG adds members outside of the “properties” member to not interfere with the content of “properties”, which is controlled by the generator of the data and may contain anything as needed for the intended use. This is a key design principle to also make information available to readers independent of how a community might represent the information in the “properties” member.

This is different in STAC and OGC API Records. Both specify an application schema, i.e., properties to be added in the “properties” container.

There was agreement that the different approaches of where to put properties are the result of the different scopes of the JSON-FG and STAC / OGC API Records specifications.

For example, if a STAC instance would (in the future) be encoded as JSON-FG, the relevant information would be duplicated in the “time”, “place”, “coordRefSys”, “featureType” members of JSON-FG.

The same is true for OGC API Records. The JSON-FG requirements class should, for now, be removed from the OGC API Records specification.

The way the information is represented may differ, but in general there is a straightforward mapping between the different representations.

However, it would be good to revisit the “..” (based on ISO 8601) vs `null` representation for unbounded interval ends,

- OGC API — Features uses `null` in JSON interval arrays in the temporal extents of a Collection resource.

- OGC API — Features uses `..` in the “datetime” query parameter since the parameter uses the ISO 8601 interval text encoding.
- CQL2 Text and CQL2 JSON currently use `".."`. Earlier this was `null` in CQL2 JSON, but was changed to align with the CQL2 Text.
- JSON-FG also used `null`, but changed to `".."` to align with CQL2 JSON.

This should be re-discussed in the Features API SWG before finalizing CQL2. Options include the following.

- Always use `null` in JSON. A `NULL` keyword could also be added in CQL2 Text. The “datetime” query parameter uses the ISO 8601 text interval encoding with `/` as the separator and therefore also `...`
- Always use `null` in JSON responses (Collection resource, JSON-FG), but continue to use `".."` in filters (i.e., CQL2).
- Leave as is.

6.5. Filtering associations with CQL2

Links and associations are a key part of a record in OGC API Records and of a STAC item. To properly support filtering on “links” or “assets” members, the CQL2 grammar needs to be extended.

- There was agreement to work on an extension to CQL2 that supports queryables that have objects and arrays as values. CQL2 currently has support for arrays, but only with simple values (string, number, boolean).
- This will be required for “links”, but will also be needed for other cases including querying the “theme” property in Records or the “assets” member in STAC.
- The property with structure like “links”, “theme” or “assets” would be published as a queryable with their schema.
- This is complementary to the approach to define a queryable that is not directly represented as a feature property in the response to make filtering data structures like “assets” simpler as described in OGC API — Features — Part 3: Filtering.
- The OGC API Records SWG should identify the requirements from the perspective of OGC API Records. Afterwards, the discussion should be moved to the Features API where the CQL2 extension should be drafted.

7

CONCLUSIONS

7 CONCLUSIONS

This was the first hybrid code sprint (consisting of both in-person and remote elements) organized by the OGC in more than two years, due to the pandemic. A record number of participants registered to attend the code sprint, exceeding pre-pandemic registration numbers. There were, however, more remote participants than those attending in-person. This suggests that there continues to be significant interest in code sprints, and that the online collaboration environment should continue to be used post-pandemic.

The code sprint facilitated the development and testing of prototype implementations of OGC and ISO Standards that relate to geospatial metadata and catalogues. The code sprint also enabled the participating developers to provide feedback to the editors of candidate standards. The code sprint therefore met all of its objectives and achieved its goal of accelerating the support of open geospatial standards that relate to geospatial metadata and catalogues.

7.1. Future Work

The sprint participants made the following recommendations for future innovation work items:

- Initiatives to facilitate implementation of JSON-FG (e.g., 3D, cadastral data, etc);
- Initiatives to facilitate implementation of catalogues; and
- Prototyping of tools for creating metadata (e.g., the automated STAC metadata crawler demonstrated during the sprint),

The sprint participants also made the following recommendations for things that the SWGs should consider:

- Outreach for JSON-FG;
- Code Sprint for designing profiles of JSON-FG for different communities of interest;
- Documentation of the different roles of catalogues and API, as well as guidance on when to use them;
- Code Sprint on versioning, possibly combining an OGC API — Features — Part 4 with OGC API — Records; and
- Exploring how to move GeoDCAT forward within OGC.

It is envisaged that the SWGs will consider the above-listed recommendations for future work items.

A

ANNEX A (INFORMATIVE) REVISION HISTORY

A ANNEX A (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2022-09-26	0.1	G. Hobona	all	initial version
2022-10-17	0.2	G. Hobona	all	r1 version

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Vretanos, P.A, Kralidis, T., Heazel, C.: OGC 20-004: Draft OGC API — Records — Part 1: Core, http://docs.ogc.org/DRAFTS/20-004.html

[2] Portele, C., Vretanos, P.A: OGC 21-045: OGC Features and Geometries JSON — Part 1: Core, https://docs.ogc.org/DRAFTS/21-045.html

[3] STAC Community: SpatioTemporal Asset Catalog, https://stacspec.org/en

[4] Holmes, C.: Static SpatioTemporal Asset Catalogs in Depth, Available at https://medium.com/radiant-earth-insights/static-spatiotemporal-asset-catalogs-in-depth-710530934a84