OGC® DOCUMENT: 23-059
External identifier of this OGC® document: http://www.opengis.net/doc/PER/oscs-2023

Open Geospatial Consortium

2023 OPEN STANDARDS CODE SPRINT SUMMARY ENGINEERING REPORT

ENGINEERING REPORT
Implementation

PUBLISHED

Submission Date: 2023-11-20
Approval Date: 2024-01-25
Publication Date: 2024-04-18
Editor: Gobe Hobona, Joana Simoes

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

License Agreement

Use of this document is subject to the license agreement at https://www.ogc.org/license

Copyright notice

Copyright © 2024 Open Geospatial Consortium
To obtain additional rights of use, visit https://www.ogc.org/legal

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

I. EXECUTIVE SUMMARY viii

II. KEYWORDS ix

III. SUBMITTERS x

IV. ABSTRACT xiii

1. INTRODUCTION 15

2. TERMS, DEFINITIONS AND ABBREVIATED TERMS 17
2.5. Abbreviated terms 18

3. HIGH-LEVEL ARCHITECTURE 20
3.1. Approved OGC Standards 20
3.1.1. OGC SensorThings API 20
3.1.2. OGC STAplus — an extension of the OGC SensorThings API 21
3.1.3. OGC API — Features 21
3.1.4. OGC API — Tiles 21
3.1.5. OGC API — Environmental Data Retrieval 21
3.1.6. OGC API — Processes 22
3.1.7. OGC CityGML 22
3.1.8. OGC CDB 22
3.1.9. OGC GeoPackage 22
3.1.10. OGC GML in JPEG 2000 (GMLJP2) 23
3.2. Candidate OGC Standards 23
3.2.1. Model for Underground Data Definition and Integration (MUDDI) 23
3.2.2. OGC API — Records 23
3.2.3. OGC API — Coverages 23
3.2.4. OGC Publish/Subscribe extension of OGC API Standards 24
3.2.5. OGC Features and Geometries JSON (JSON-FG) 24
3.3. Specifications from the Community 24
3.3.1. NGIIS GIMI 24
3.3.2. AsyncAPI 25
3.4. Software Projects and Products 26
3.4.1. OSGeo pygeoapi 26
3.4.2. OSGeo pycsw 26
3.4.3. ldproxy 26
3.4.4. MariaDB Geospatial Data Server 26
3.4.5. GNOSIS Map Server 27

3.4.6. web-gimi-viewer 27

4. RESULTS 30
4.1. Approved and Candidate OGC API Standards 30
4.1.1. OGC API — Records 30
4.1.2. OGC API — Environmental Data Retrieval 33
4.1.3. OGC API — Features 34
4.1.4. OGC API — Maps 36
4.1.5. STAplus SensorThings API Extension 36
4.2. Approved and Candidate OGC Encoding Standards 43
4.2.1. OGC Features and Geometries JSON 43
4.2.2. OGC CDB 47
4.2.3. GML and JSON-FG schemas for the MUDDI Core Logical Model 47
4.2.4. MUDDI logical schema experimentation 48
4.2.5. Analysis of MUDDI by IIT Bombay 51
4.3. Products applied to OGC Standards in this code sprint 55
4.3.1. MariaDB 55
4.3.2. ldproxy 92
4.3.3. OSGeo pycsw 97
4.3.4. Safe Software FME 97
4.3.5. Hexagon Catalog Explorer 105
4.3.6. Geonovum testing of JSON-FG support in GDAL 106
4.3.7. OSGeo pygeoapi 107
4.4. Products applied to OGC Standards and GIMI in this code sprint 110
4.4.1. pygeoapi support for the draft GIMI profile 110
4.4.2. Storage of GIMI files through a Web API 113
4.4.3. Towards open source libraries for GIMI 117
4.4.4. Testing Windows open source support for GIMI 124
4.4.5. GIMI web-map client 126
4.4.6. Ecere GNOSIS Map Server 132
4.4.7. GIMI support in the MiraMon Web Browser 134

5. DISCUSSION 140
5.1. JSON-FG 140
5.2. MUDDI 140
5.3. OGC API Standards 141
5.3.1. Towards a common Publish/Subscribe Event Driven Approach 141
5.3.2. Security 141
5.3.3. Schemas 141
5.4. GIMI 142

6. CONCLUSIONS 144
6.1. Future Work 145
6.1.1. OGC API Standards 145
6.1.2. OGC Encoding Standards 145
6.1.3. GIMI 146

ANNEX A (INFORMATIVE) REVISION HISTORY 148

BIBLIOGRAPHY 150

LIST OF TABLES

Table — Submitters x
Table 1 — Mapping of HEIF media types to OGC API content negotiation 133
Table — Revision History 148

LIST OF FIGURES

Figure 1 — High Level Overview of the Sprint Architecture 20
Figure 2 — The STAplus and GeoXACML based architecture implemented by the CitiObs project 38
Figure 3 — Fragment from example GeoXACML policy 39
Figure 4 — STAplus Entity Type diagram 40
Figure 5 — Screenshot of the interface when a user is NOT logged into the TAPIS system 41
Figure 6 — Screenshot of the interface showing observations when a user has been authenticated 42
Figure 7 — Screenshot of the interface when authenticated user is prevented from accessing other user's observations 42
Figure 8 — Screenshot of the json-fg-validator tool providing an ETS report. 44
Figure 9 — Illustration of a MUDDI logical schema for a relational database 49
Figure 10 — Relationships between types of networks in MUDDI 50
Figure 11 — A screenshot of a web application displaying underground asset data 51
Figure 12 — Screenshot of MUDDI data displayed in FME Data Inspector 52
Figure 13 — Comparison of a sample of feature properties from gas provider data 53
Figure 14 — Comparison of a sample of feature properties from electricity provider data 54
Figure 15 — A sample of MUDDI-harmonized data uploaded to a PostGIS database and displayed on QGIS 55
Figure 16 — MariaDB's Geospatial GeoPortlet listing layers created from MUDDI conformant data 57
Figure 17 — Layers displayed on a map on MariaDB's Geospatial GeoPortlet 58
Figure 18 — Accessing a layer through the Features API 59
Figure 19 — Screenshots of the TEP accessing the catalog to locate images of interest 89
Figure 20 — Selection of an image and compatible process 91

Figure 21 — Screenshots of process execution (top-left), job acceptance (top-right), job status (lower-left), and display of result (lower-right) 92
Figure 22 — ldproxy landing page 94
Figure 23 — ldproxy accessed from MQTT Explorer 96
Figure 24 — Screenshot of FME workflow for converting Burnaby underground data from plain GeoJSON to MUDDI GML 98
Figure 25 — Screenshot of MUDDI GML data displayed in FME for a NetworkConveryanceLink feature. Note the reference to the associated node xlink_href. Also note the 3D geometry with depth stored as z values on the feature coordinates. 100
Figure 26 — Screenshot of FME workflow for converting from MUDDI GML to MUDDI compliant GeoJSON, GeoPackage and Geodatabase. 101
Figure 27 — Screenshot of MUDDI GeoJSON data displayed in FME. Note the simplified schema with empty properties not shown. 102
Figure 28 — Screenshot of MUDDI GeoPackage data displayed in FME showing a NetworkConveyanceNode feature. Note that the gml_id of this node matches the hasInlet.xlink_href of the connected NetworkConveyanceLink feature shown in the MUDDI GML output above. 102
Figure 29 — Screenshot of workflow for converting from CityGML2 to CityGML3 103
Figure 30 — Screenshot of workflow for converting from IFC to CityGML3 104
Figure 31 — Screenshot of the output CityGML data displayed in FME 104
Figure 32 — Screenshot of the Hexagon Catalog Explorer accessing an implementation of OGC API - Records 106
Figure 33 — Screenshot of the pygeoapi OGC API - Records provider architecture for the CSW Facade. 107
Figure 34 — Screenshot of the pygeoapi OGC API - Records provider for the CSW Facade. 108
Figure 35 — Screenshot of MQTT Explorer with metadata notification updates. 109
Figure 36 — Screenshot of metadata published to OGC API - Records using pygeoapi. 109
Figure 37 — Screenshot of pygeoapi's HTML interface, showing a GIMI coverage collection. The collection's geographic bounds are displayed by pygeoapi. 111
Figure 38 — Screenshot of the leaflet client consuming tiles, which are served by pygeoapi, in the PNG format, and are generated from the GIMI file. 112
Figure 39 — Screenshot showing an image sent using postman (right hand-side), and stored (lower-left), through a process in pygeoapi (upper-left) 116
Figure 40 — Illustration of the ecosystem 119
Figure 41 — Screenshot of a test GIMI image displayed in QGIS 120
Figure 42 — Screenshot of a jupyter notebook for displaying a GIMI file 121
Figure 43 — Second screenshot of a jupyter notebook for displaying a GIMI file 122
Figure 44 — Overview of bounds, domain set and ranges of a coverage from a GIMI file on pygeoapi 123
Figure 45 — GIMI image file displayed in OpenLayers 124
Figure 46 — Code change in Python script for parsing GIMI file with pyisobmff 125
Figure 47 — Using pyisobmff for reading GIMI image 126

Figure 48 — Schematic data flow of server-side GIMI processing for display 127
Figure 49 — Schematic data flow of client-side GIMI processing for display 128
Figure 50 — Metadata encodings found in a GIMI file 129
Figure 51 — Screenshot of the prototype web-gimi-viewer displaying metadata from a GIMI file in text form 130
Figure 52 — Screenshot of the prototype web-gimi-viewer displaying security and TAI metadata from a GIMI file 131
Figure 53 — Screenshot of the prototype web-gimi-viewer displaying a GIMI file with HEIF imagery. Contains modified Copernicus Sentinel data 2023. 132
Figure 54 — Screenshot of the web interface showing the data and image extracted from the heif file 135
Figure 55 — Screenshot of the MiraMon Map Browser showing a HEIF image from an OGC API Maps. 136
Figure 56 — Screenshot of the MiraMon Map Browser showing the histogram of a HEIF image from OGC API maps. 137
Figure 57 — Screenshot of the MiraMon Map Browser showing the calculator of layers with an HEIF image from OGC API maps. 137
Figure 58 — Screenshot of the MiraMon Map Browser showing the result of a calculation using a HEIF image from OGC API maps. 138

I EXECUTIVE SUMMARY

Advances in computing, storage, sensing, and other technology areas continue to find a place in the geospatial ecosystem that makes location information findable, accessible, interoperable, and reusable. To enable geospatial standards to meet the likely interoperability needs of future technologies, it is necessary for Standards Development Organizations (SDOs) such as OGC to continue to innovate. Such innovation relies on the ability to develop new standards and improve existing ones to better address emerging interoperability challenges. Code sprints are one of a number of initiative types that OGC conducts to accelerate such innovation. 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 candidate standards.

The focus of this Engineering Report (ER) is a code sprint that was held from October 30th to November 1st, 2023. The code sprint was organized by the Open Geospatial Consortium (OGC) and hosted by Geovation in London, England. The code sprint was sponsored by Ordnance Survey (OS) at the Gold-level, as well as the European Union Satellite Centre (SatCen), US National Geospatial Intelligence Agency (NGA), and the UK Defence Science and Technology Laboratory (Dstl) at the Silver-level. As with previous OGC Code Sprints, a key goal of the October 2023 Open Standards Code Sprint was the acceleration of the development of OGC Standards and their implementation in software products.

Standards in focus for this code sprint included several OGC API, data model, and encoding standards. In addition to OGC Standards, the code sprint also included a thread focusing on the "GEOINT Imagery Media for ISR" (GIMI) profile of the ISO Base Media File Format (ISOBMFF) and the High Efficiency Image Format (HEIF). The engineering report concludes that the code sprint met all of its objectives and achieved its goal of accelerating the development of OGC Standards and their implementation in software products.

In relation to emerging extensions to OGC API Standards, the engineering report concludes that the Publish/Subscribe (PubSub), Security, and Schemas capabilities examined in this code sprint could potentially support other OGC API Standards. Regarding GIMI, the engineering report concludes that with minor changes to popular open-source base libraries a wide range of software can quickly make use of GIMI capabilities.

Regarding the Model for Underground Data Definition and Integration (MUDDI), the engineering report concludes that the MUDDI Conceptual Model is implementable and could easily be used as the basis for a logical model that can be encoded in a variety of formats including GML, GeoJSON, and Geopackage. As for JSON-FG, the engineering report concludes that due to the evident increase in adoption across the developer community, there is an increasing need to offer validators so that developers can check for compliance more easily.

The sprint participants made the following recommendations for future work items on OGC API Standards.

- Extension of OGC API — Records to support faceting and federated search.

- Use case experimentation to examine how the Publish/Subscribe approach can support various OGC API standards workflows.
- Prototyping and specification on an OpenAPI definition document for the OGC SensorThings API Standard.
- Additional support for multiband raster data in OGC API — Maps/Tiles/Coverages (e.g., multispectral imagery, etc).

The sprint participants made the following recommendations for future work items on OGC Encoding Standards.

- Tiled complex, signed, and floating-point data in multiband raster data.
- Performance benchmarking between various imagery formats, for a variety of use cases.
- Extraction of the conceptual model of GMLJP2 to make it applicable to any ISOBMFF profile.
- Development of an Implementation Standard for the logical model and encoding of data that conforms to the MUDDI.
- Use of the MUDDI logical schema as a template for MUDDI compliance.

The sprint participants made the following recommendations for future work items on GIMI.

- Extension of open-source base libraries used by several other products for creation and modification of GIMI files.
- Prototype streaming of large images from client applications to servers for storage through OGC API — Processes.
- Prototyping of a JavaScript Object Notation (JSON) encoded alternative to Key-Length-Value (KLV) metadata for use in GIMI files.

II KEYWORDS

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

ogcdoc, OGC document, API, openapi, html, underground, json-fg, nuar, gimi, heif, muddi

III SUBMITTERS

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

Table — Submitters

NAME	ORGANIZATION	ROLE
Gobe Hobona	OGC	Editor
Joana Simoes	OGC	Editor
Tom Kralidis	Meteorological Service of Canada	Editor
Andreas Matheus	Secure Dimensions	Contributor
Iván Sánchez Ortega	OSGeo	Contributor
Florent Gravin	Camptocamp	Contributor
Clemens Portele	interactive instruments GmbH	Contributor
Johannes Echterhoff	interactive instruments GmbH	Contributor
Jérôme Jacovella-St-Louis	Ecere Corporation	Contributor
Brad Hards		Contributor
Sine Kelly	Esri UK Ltd	Contributor
Rebecca Aspin	Ordnance Survey	Contributor
Miles Austen	Ordnance Survey	Contributor
Nikita Dabizha	VOLTQUANT	Contributor
George Kolokotronis	VOLTQUANT	Contributor
Mete Ercan Pakdil	Mott MacDonald	Contributor
Núria Julià Selvas	CREAF	Contributor

NAME	ORGANIZATION	ROLE
Satish Sankaran	Esri	Contributor
Jonathan Bates	GeoPlace LLP	Contributor
Conor Lambert	Ordnance Survey	Contributor
Ricardo Silva	Geobeyond	Contributor
Arnoud de Boer	Geonovum	Contributor
Goncalo Lopes	NeuroGEARS Ltd	Contributor
Joe Guyatt	Ordnance Survey	Contributor
Rob Smith	Away Team Software	Contributor
James Clarke	Ordnance Survey	Contributor
Richard Hargreaves	Ordnance Survey	Contributor
Neil Kirk	Envitia Ltd	Contributor
Alex Scott-Johns	Envitia Ltd	Contributor
Dave Rowe	Ordnance Survey	Contributor
Carsten Roensdorf	Ordnance Survey	Contributor
Simon Smith	Ordnance Survey	Contributor
Chris Popplestone	Ordnance Survey	Contributor
Neil Brammall	Geospatial Commission	Contributor
Gareth Boyes	MGISS	Contributor
Michael Gordon	Ordnance Survey	Contributor
Colin Sawkins	Cadent Gas Limited	Contributor
Matthew Lewis	Esri UK	Contributor

NAME	ORGANIZATION	ROLE
Sumit Sen	GISE Hub, IIT Bombay	Contributor
Paul Hann	Esri UK	Contributor
Joost Farla	Geonovum	Contributor
Samantha Lavender	Pixalytics Ltd	Contributor
Panagiotis (Peter) Vretanos	MariaDB / CubeWerx	Contributor
Lucio Colaiacomo	EU Satellite Centre	Contributor
Scott Simmons	OGC	Contributor
Chris Little	Met Office	Contributor
Michael Leedahl	Maxar	Contributor
Ashiraf Kyabainze	AT HAUSE LIMITED	Contributor
Linda van den Brink	Geonovum	Contributor
Ingo Simonis	OGC	Contributor
Thomas Hodson	ECMWF	Contributor
Steven McDaniel	Hexagon	Contributor
Charla Harbin	Hexagon	Contributor
Joan Maso	UAB-CREAF	Contributor
Iain Burnell	Dstl	Contributor
Bryan Seeds	Dstl	Contributor
Christopher Budas	Dstl	Contributor
Simon Knight	Dstl	Contributor
Amy Baker	Dstl	Contributor

NAME	ORGANIZATION	ROLE
Devon Sookhoo	TRAX International Corporation	Contributor
Joe Stufflebeam	TRAX International Corporation	Contributor
Dean Hintz	Safe Software	Contributor
Ryan Franz	FlightSafety International	Contributor
Hylke van der Schaaf	Fraunhofer Gesellschaft	Contributor

IV

ABSTRACT

The subject of this Engineering Report (ER) is a code sprint that was held from October 30th to November 1st, 2023 to advance the support and development of open standards within the developer community. The code sprint was organized by the Open Geospatial Consortium (OGC) and hosted by Geovation in London, England. The code sprint was sponsored by Ordnance Survey (OS), the European Union Satellite Centre (SatCen), US National Geospatial Intelligence Agency (NGA), and the UK Defence Science and Technology Laboratory (Dstl).

1

INTRODUCTION

1 INTRODUCTION

Code sprints are one of a number of initiative types that OGC holds during the course of each year. Other types of initiatives are design experiments, pilots, plugfests, operational exercises, interoperability experiments, and testbeds. OGC Code sprints can be either 'universal' or 'focused'. All OGC working groups are invited and encouraged to set up a thread in a universal code sprint, whereas a focused code sprint is tailored to limit the number of standards in-focus to approximately three.

The October 2023 Open Standards Code Sprint was held as a universal code sprint meaning that all OGC working groups were encouraged to participate. As a result, several OGC Standards Working Groups (SWGs) set up teams of developers to collaborate during the three-day event. Whilst individual developers work on specific standards during the code sprint, these developers also get the opportunity to interact and discuss with other developers working on other standards. This makes code sprints a rich environment for knowledge transfer across teams, as well as for nurturing cross-functional teams.

This ER presents the high-level architecture of the code sprint and describes each of the standards and software packages that were deployed in support of the code sprint. The ER also discusses the results and presents a set of conclusions and recommendations. The recommendations identify ideas for future work, some of which may be more appropriate for testbeds, pilots, or other types of OGC initiatives. Therefore, the reader is encouraged to consider the recommended future work within the context of all OGC Standards development, collaborative solutions, and innovation activities.

2

TERMS, DEFINITIONS AND ABBREVIATED TERMS

2 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.

2.1. 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).

2.2. coordinate reference system

A coordinate system that is related to the real world by a datum term name (source: ISO 19111).

2.3. 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).

2.4. **Web API**

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

2.5. Abbreviated terms

API	Application Programming Interface
CITE	Compliance Interoperability & Testing Evaluation
CRS	Coordinate Reference System
EDR	Environmental Data Retrieval
GIMI	GEOINT Imagery Media for ISR (GIMI)
GIS	Geographic Information System
MUDDI	Model for Underground Data Definition and Integration
NGIIS	Next Generation ISR Imagery Standards
OGC	Open Geospatial Consortium
OWS	OGC Web Services
REST	Representational State Transfer
TEAM	Test, Evaluation, And Measurement Engine

3

HIGH-LEVEL ARCHITECTURE

3 HIGH-LEVEL ARCHITECTURE

The focus of the code sprint was on the support of implementations of open geospatial standards across various software projects. Implementations of approved and candidate OGC Standards were deployed in participants' own infrastructure in order to build an environment with the architecture shown below in Figure 1. As illustrated, the sprint architecture was designed to enable client applications to connect to different servers that implement a variety of standards. The architecture also included several different software libraries that support open geospatial standards and enable the extraction, transformation, and loading of geospatial data.

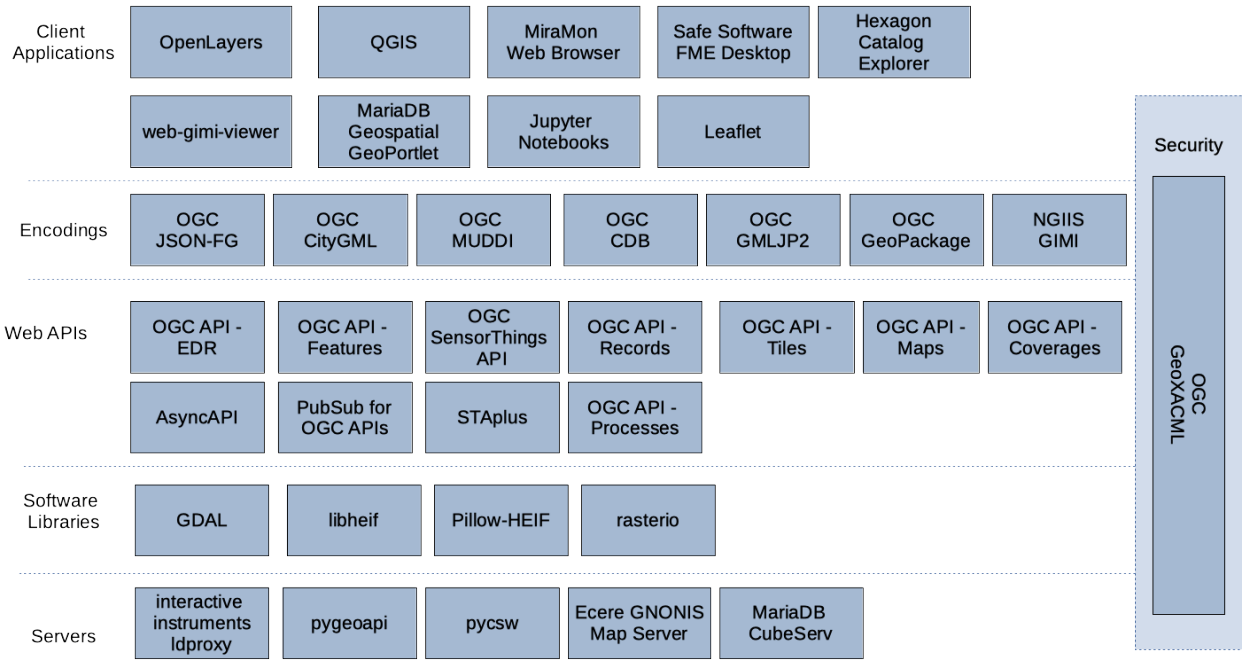


Figure 1 — High Level Overview of the Sprint Architecture

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

3.1. Approved OGC Standards

3.1.1. OGC SensorThings API

The OGC SensorThings API Standard provides an open and harmonized way to interconnect devices, applications, and data over the web and on the Internet of Things (IoT) (OGC 18-088). At a high level the SensorThings API provides two main parts, namely Part I — Sensing, and Part II — Tasking. The Sensing part of the Standard provides a way to manage and retrieve

observations and metadata from different sensor systems. The Tasking part of the Standard provides a way for tasking IoT devices, such as actuators and sensors. The SensorThings API follows REST principles and uses JSON for encoding messages as well as Message Queuing Telemetry Transport (MQTT) for publish/subscribe operations.

3.1.2. OGC STAplus — an extension of the OGC SensorThings API

STAplus is an extension to the OGC SensorThings data model (OGC 22-022r1). Inspired by Citizen Science applications, STAplus supports the 'ownership concept' whereby observations collected by sensors are owned by (different) users that may express the license for re-use. In addition to the ownership and license abilities, the extension specified by STAplus supports the expression of explicit relations between observations and the creation of group(s) of observations.

3.1.3. OGC API — Features

The OGC API — Features Standard offers the capability to create, manage, and query spatial data on the Web. The Standard specifies requirements and recommendations for Web APIs that are designed to facilitate the sharing of feature data. The specification is a multi-part standard. Part 1, labelled the Core, describes the mandatory capabilities that every implementing service has to support and is restricted to read-access to spatial data that is referenced to the World Geodetic System 1984 (WGS 84) Coordinate Reference System (CRS) (OGC 17-069r4). Part 2 enables the use of different CRSs, in addition to the WGS 84 (OGC 18-058r1). Additional capabilities that address specific needs will be specified in additional parts. Envisaged future capabilities include, for example, support for creating and modifying data, more complex data models, and richer queries.

3.1.4. OGC API — Tiles

OGC API — Tiles specifies a Standard for Web APIs that provide tiles of geospatial information (OGC 20-057). The Standard supports different forms of geospatial data, such as tiles of vector features (colloquially called "vector tiles"), coverages, maps (or imagery), and potentially eventually additional types of tiles of geospatial data.

Vector data represents geospatial objects such as points, lines, and polygons. Tiles of vector feature data (i.e., 'vector tiles') represent partitions of vector data covering an area (e.g., lines representing rivers in a country).

In this context, a map is essentially an image representing at least one type of geospatial information. Tiles of maps (i.e., map tiles) represent subsets of maps covering an area.

3.1.5. OGC API — Environmental Data Retrieval

The OGC API — Environmental Data Retrieval (EDR) Standard provides a family of lightweight interfaces to access Environmental Data resources. Each resource addressed by an EDR API maps to a defined query pattern. This Standard identifies resources, captures compliance classes,

and specifies requirements which are applicable to OGC Environmental Data Retrieval API's. This Standard addresses both discovery and query operations. Discovery operations enable the API to be interrogated to determine its capabilities and retrieve metadata about the published resource. Query operations allow Environmental Data resources to be retrieved from the underlying data store based upon simple selection criteria, defined by this standard and selected by the client.

Version 1.1 of OGC API — EDR has recently been published (OGC 19-086r6). The EDR API Standards Working Group (SWG) has since commenced work on Part 2 of the Standard, which is tentatively titled "OGC API — Environmental Data Retrieval — Part 2: Publish-Subscribe workflow" (OGC 23-057). The focus of the EDR API-related work in this code sprint was therefore on Part 2 of the Standard.

3.1.6. OGC API — Processes

The OGC API — Processes Standard supports the wrapping of computational tasks into executable processes that can be offered by a server through a Web API and be invoked by a client application (OGC 18-062r2). The standard enables the execution of computing processes and the retrieval of metadata describing the purpose and functionality of the processes. Typically, these processes execute well-defined algorithms that ingest vector and/or coverage data to produce new datasets.

3.1.7. OGC CityGML

The CityGML Standard specifies a conceptual model and encoding format for the representation, exchange, and storage of virtual 3D city models (OGC 21-006r2). The standard facilitates the integration of urban geodata for a variety of use cases, for example Urban Digital Twins, Smart Cities, Building Information Modeling (BIM), transportation, and environmental management.

3.1.8. OGC CDB

The CDB Standard defines a standardized model and structure for a single virtual representation of the earth. A CDB structured data store provides for a geospatial content and model definition repository that is plug-and-play interoperable between database authoring workstations. The CDB Standard defines an open format for the storage, access, and modification of a database, typically, but not exclusively, a synthetic environment database used for computer simulation.

3.1.9. OGC GeoPackage

The GeoPackage Encoding Standard specifies the rules and requirements for an open, standards-based, platform-independent, portable, self-describing, compact format for transferring geospatial information (OGC 12-128r18). The GeoPackage standard defines the schema for a GeoPackage, including table definitions, integrity assertions, format limitations, and content constraints.

3.1.10. OGC GML in JPEG 2000 (GMLJP2)

The OGC's GML in JPEG 2000 for Geographic Imagery Encoding Standard (GMLJP2) defines the means by which the Geography Markup Language (GML) Standard is used within JPEG 2000 images for geographically referenced imagery (OGC 08-085r8). The primary role of GMLJP2 is to geographically reference imagery placed in JPEG 2000 files which is achieved through description of the image geometry (domainSet) and radiometry (rangeType) by using the OGC's Coverage Implementation Schema (CIS) Standard. GMLJP2 makes use of a built-in ability of JPEG 2000 files to include XML data for description of an image within a JPEG 2000 data file.

3.2. Candidate OGC Standards

3.2.1. Model for Underground Data Definition and Integration (MUDDI)

The Model for Underground Data Definition and Integration (MUDDI) candidate Standard serves as a framework to make datasets that utilize different information for underground objects interoperable, exchangeable, and more easily manageable.

The MUDDI Conceptual Model represents real-world objects found underground. It was designed as a common basis to create implementations that make different types of subsurface data — such as those relating to utilities, transport infrastructure, soils, ground water, or environmental parameters — interoperable in support of a variety of use cases and in different jurisdictions and user communities.

3.2.2. OGC API — Records

The OGC API — Records candidate Standard provides discovery and access to metadata records that describe resources such as features, coverages, tiles / maps, models, assets, datasets, services, or widgets (OGC 20-004). 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.

3.2.3. OGC API — Coverages

The OGC API — Coverages candidate standard establishes an access mechanism for coverages as defined by the OGC Abstract Specification Topic 6 Schema for coverage geometry and functions (OGC 20-058). The mechanism is established through a Web API which can be described by an API description language such as the OpenAPI specification. Arguably the most popular type of coverage is that of a gridded coverage. Gridded coverages, such as raster

imagery, have a grid as their domain set describing the direct positions in multi-dimensional coordinate space, depending on the type of grid.

3.2.4. OGC Publish/Subscribe extension of OGC API Standards

OGC Web APIs provide capabilities that are typically based on HTTP requests initiated by clients. Clients have to poll APIs for collection resource updates (new features/records items, coverages, maps, etc.). Depending on a collection's temporal resolution or frequency of updates, an event-driven / Publish-Subscribe architecture provides a timely, efficient, and low latency approach for delivery of data updates.

3.2.5. OGC Features and Geometries JSON (JSON-FG)

The draft OGC Features and Geometries JSON (JSON-FG) Standard extends the GeoJSON format 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 (OGC 21-045). In particular, the JSON-FG Standard specifies the following extensions to the GeoJSON format:

- the ability to use Coordinate Reference Systems (CRSs) other than WGS 84;
- support for solids and prisms as geometry types;
- the ability to encode temporal characteristics of a feature; and
- the ability to declare the type and the schema of a feature.

3.3. Specifications from the Community

3.3.1. NGIIS GIMI

Next Generation ISR Imagery Standards (NGIIS) is a program of work to modernize Intelligence, Surveillance, and Reconnaissance (ISR) for still and motion imagery standards. Key priorities are:

- leverage open commercial media standards to integrate still and motion imagery into a single container;
- harmonization of still and motion imagery metadata through development of an imagery domain ontology;
- address a broad range of use cases with a common, integrated suite of capabilities;
- enable automation; and
- support to disadvantaged and remote users.

NGA Standard 0076 (NGA.STND.0076) “GEOINT Imagery Media for ISR” (GIMI), currently in draft, defines a profile of the International Standards Organization / International Electrotechnical Commission (ISO/IEC) 14496-12 ISO Base Media File Format (ISOBMFF) video/audio standard and the ISO/IEC 23008-12 High Efficiently Image File Format (HEIF) Still Imagery standard. The GIMI Profile is a central element of the NGIIS program. GIMI is expected to replace the National Imagery Transmission Format (NITF) that is widely used within the US Department of Defense for ISR. It is also the likely replacement for STANAG 4545 NATO Secondary Imagery Format (NSIF) with NATO, given the close ties between NITF and NSIF.

GIMI is also expected to replace use of MPEG-2 Transport Stream in most applications using motion imagery.

GIMI builds on top of the HEIF and ISOBMFF standards. Key additions include security metadata, geopositioning and other per-image metadata and a globally-unique content identifier.

The security metadata provides a way to label the file with overall security classification, security caveats, countries to which the data may be released, and declassification information. This would typically be in the form of a US Intelligence Community marking standard known as Information Security Marking (ISM), which makes use of XML. NATO marking may use STANAG 4774 and STANAG 4778 standards. Other countries may choose a different data model and encoding, depending on national policies. For the purpose of the sprint, an indicative but imaginary format was used. Use of fake markings avoided any possible perception that test files were actually security classified data.

The final general-purpose metadata for GIMI is still in development, but is likely to be a form of Resource Description Format (RDF), likely encoded in SMTPE ST 336 Key-Length-Value (KLV) for efficiency. For the purpose of the sprint, the MISB ST 0601 UAS KLV standard was used to provide this metadata. The primary metadata was the geospatial location of the corner points of each image. There is no expectation that ST 0601 will form part of the final GIMI metadata solution, but ST 0601 KLV is possibly the best known and most widely implemented KLV encoding and provided enough information to support image geopositioning.

The Content ID was provided as a UUID encoded as a URN.

3.3.2. AsyncAPI

AsyncAPI is a specification by the Linux Foundation for describing and documenting message-driven APIs in a machine-readable and protocol-agnostic format. The specification defines a set of files required to describe a message-driven API. The files describing an API in accordance with the AsyncAPI Specification may be encoded in JSON or YAML.

3.4. Software Projects and Products

3.4.1. 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.

3.4.2. OSGeo pycsw

pycsw is an implementation of the OGC API — Records candidate Standard and the OGC's Catalog Service for the Web (CSW) Standard 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 catalog 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 the OGC API — Records — Part 1: Core, version 1.0 candidate Standard and its associated specifications. The official documentation provides an overview of all supported standards.

3.4.3. ldproxy

ldproxy is an implementation of the OGC API family of Standards, available under the MPL 2.0 open source license. ldproxy is developed by interactive instruments GmbH, written in Java, and is deployed using Docker containers. ldproxy implements all parts of OGC API — Features, OGC API — Tiles, OGC API — Styles, OGC API — 3D GeoVolumes, and OGC API — Routes. ldproxy is an OGC Reference Implementation for Parts 1 and 2 of OGC API — Features.

3.4.4. MariaDB Geospatial Data Server

The MariaDB Geospatial Data Server ("cubeserv") is implemented in C and currently implements the following OGC Standards and draft specifications.

- Multiple conformance classes and recommendations of the OGC API — Tiles — Part 1: Core Standard
- Multiple conformance classes and recommendations of the OGC API — Maps — Part 1: Core candidate Standard

- 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, for example, Geography Markup Language (GML), GeoJSON, Mapbox Vector Tiles, MapMP, as well as several coordinate reference systems.

3.4.5. GNOSIS Map Server

The GNOSIS Map Server is written in the eC programming language and supports multiple OGC API Standards. GNOSIS Map Server supports multiple encodings including GNOSIS Map Tiles (which can contain either vector data, gridded coverages, imagery, point clouds, or 3D meshes), Mapbox Vector Tiles, GeoJSON, GeoTIFF, GML, and MapML. An experimental server is available online at https://maps.gnosis.earth/ogcapi and has been used in multiple OGC Innovation Program initiatives.

3.4.6. web-gimi-viewer

web-gimi-viewer is a proof-of-concept viewer for NGIIS GIMI imagery. It is designed to run entirely in a web browser, and is intended to be a stepping stone towards more refined web client implementations as well as a testing platform for the integration of all the components needed to consume GIMI imagery.

It leverages:

- mp4box for the ISOBMFF container;
- vidterra's misb.js for the ST0601 KLV metadata;
- The browser's XML parser for the security metadata;
- heic2any for the HEIC image codec; and

- Gleo for the web map interface.

4

RESULTS

4 RESULTS

The code sprint included multiple software applications and experimented with several standards. This section presents the key results from the code sprint.

4.1. Approved and Candidate OGC API Standards

4.1.1. OGC API — Records

The OGC API — Records team held two breakout sessions which focused on advancing the specification for review by the OGC Architecture Board (OAB).

Discussions included the following.

- Target of December 2023 for OGC API — Records — Part 1: Core document submission for OAB:
  - issue sweep;
  - requirements refactoring (shared query parameters);
  - resulting ATS refactoring;
  - update on templating links (allowing for using one of `variables` or `varBase`);
  - clarification on …/`items` `q` parameter; and
  - clean-up and validation of example JSON documents; agreement to tidy up YAML schemas and build CI workflow to ensure schemas are always valid, a bundle is built, and examples are schema valid.
- Proposal for facets:
  - advertising facets and their types (options: separate endpoints, addition of `facet: true` properties in (…/`queryables`);
  - specifying facets on …/`items` requests;
  - agreement to review / approve proposed Pull Request (PR) and move forward as an extension; and
  - A detailed description of the proposal is presented in Clause 4.1.1.1.
- Discussion on federation/distributed search as follows.

- How to address live / distributed search (without metadata harvesting/metadata aggregation).
- Current issues such as:
  - how to present results (multipart, single document with 'grouped' records [would break GeoJSON interoperability]); and
  - how to sort results; this could be accomplished by a front-end client (in Python or JavaScript, for example).
- Some organizations want to provide a unified single catalog that streamlines the search experience for users.
- Could also be put forth as a best practice on implementation.
- Way forward could be to put forth a proposal for discussion on requirements.

4.1.1.1. Facets extension of OGC API — Records

The code sprint was an excellent opportunity for developers to meet in person and advance the momentum on the facets proposal, which was initiated at the last OGC Code Sprint in Bussigny.

This subsection describes a proposal that aims to introduce a new extension to the OGC API — Records candidate Standard, facilitating a more straightforward understanding of dataset distribution across various criteria.

The term "Facet" here refers to a high-level piece of information computed over a set of records in a collection.

4.1.1.1.1. Work Achieved

The team successfully prepared a proposal for integrating the facets feature into the OGC API — Records candidate Standard. This proposal includes the following.

- Adding a new entry point `/facets` that returns predefined facets for the record. The concept is that the server administrator predefines a pool of facets that the client can use to represent the dataset distribution over certain fields.
- Specifying the different types of facets, their characteristics, and settings, including adding new parameters to the `/items` entry point to fetch facets alongside search results.

4.1.1.1.2. Definition of a Facet

A facet can be of several types as follows.

- Terms facet: A `terms` facet can be applied to any text property and produces a list of values appearing for a specific property across all matching records, as well as the count of records containing each value.

A terms facet based on `keyword` countries might return a list of buckets like so:

```
Greece (24 records)
Germany (12 records)
England (5 records)
```

- Histogram facet: A `histogram` facet can be applied to any temporal or numeric property to distribute item values over ranges or intervals.

The facet `createDate` will return a list of buckets like so:

```
2020-01-01 to 2020-02-01: 18 records
2020-02-01 to 2020-03-01: 22 records
2020-03-01 to 2020-04-01: 43 records
```

- Filters facets: A `filters` facets produces a count of matching records for one or several predefined queries. This essentially lets the user run "sub-queries" cheaply to have a better understanding of the composition of the search results like so.

```
hasDownloads : 2051 records
hasMaps : 11495 records
```

4.1.1.1.3. Discussions included

- Seeking validation from the OGC API — Records SWG to accept this new feature.
- Technical considerations
- Roadmap planning

4.1.1.1.4. Roadmap

- Subject to approval by the SWG, the proposal's PR should be finalized, reviewed, and merged.
- Subsequently, the team will focus on creating the extension definition resources.
- Simultaneously, an initial implementation will be carried out in GeoNetwork and pygeopi.

4.1.2. OGC API — Environmental Data Retrieval

The OGC API — Environmental Data Retrieval (EDR) team and PubSub enthusiasts provided two 5-minute pitch presentations, one mentor session, one breakout which focused on Publish-Subscribe workflows for EDR, and other OGC API specifications.

The 5-minute pitch presentations focused on Part 1: Core and Part 2: Publish-Subscribe capabilities to help position the subsequent mentor and breakout sessions.

The EDR PubSub mentor session included demonstrations from the joint Open Software and Open Standards Code Sprint between OGC/OSGeo/ASF that was held in Lausanne during the month of April 2023 (OGC 23-025), as well as a demo from NOAA demonstrating PubSub integration with EDR.

A breakout session was also held, focusing on technical details of EDR Part 2.

An ldproxy PubSub demo was given connecting to the WMO WIS2 Global Broker, triggering notification observations into an OGC API — Features collection as well as an additional ldproxy broker. Observations were made available using the OGC API endpoint structure with a prefix for a given server and suffix for a specific observed property. Security was also demonstrated by way of using OpenID Connect.

Discussions included the following.

- Links requirements as discussed in Issue 451.
- Adding a `channel` property to a link object is valuable for efficient binding capabilities for PubSub aware clients. Non-aware clients would simply be presented a link to a PubSub broker without further context, this is what the AsyncAPI document is there for (at the landing page, etc.).
- Addition of the ability to cross-reference collection identifiers from an AsyncAPI document, `channels.*.x-ogc-api-collection.url`, for example.
- Update link requirement to be a permission to facilitate PubSub aware client interoperability done in Pull Request 464.
- Channels requirements as discussed in Issue 452.
- Is the OGC API endpoint topic structure enough?
- The Testbed-19 ldproxy demonstration prefixed topics with tokens to help make the topics unique, which is valuable in the context of broker message aggregation.
- Consideration of filters as part of a topic, could be advanced functionality.
- Turn channels requirement into separate conformance class, done in Pull Request 465.
- Message payload: should make clear that if the message payload is an item, then the `id` property should persist. Else, if a notification, then the `id` property would be a GUID.

- Cross SWG engagement and longer-term PubSub SWG activities. PubSub is a valuable capability for numerous OGC APIs:
    - Connected Systems
    - Features: feature transactions
    - Processes: subscribing to async jobs channels, new processes
    - Records: aggregation/metadata ingest/publication
    - Common: collections publishing data updates (e.g. publication of new NWP model run for EDR or Maps)
- Engage with SWGs to determine PubSub approach to notify workflow/patterns on changes to resources.

A live demo was given which exemplified the capabilities of PubSub as a workflow for metadata catalogue management.

4.1.3. OGC API — Features

The OGC API — Features team held a breakout to review the initial version of the new Part 5: Schemas.

The general approach taken by the new part was supported. Several proposals for changes were identified during the review as follows.

- Change `/req/schemas/logical-model` into informative text (the requirement is not testable).
- Add a “x-ogc-property-index” keyword in “Schemas” to express the sequence of the properties (in a coverage or for a feature encoding that is ordered, e.g. GML).
- Only reference JSON Schema 2020-12 (latest version of JSON Schema).
- In `/rec/schemas/properties` clarify that no other keywords should be used.
- Add “x-ogc-unit” as a keyword in “Schemas” to declare the unit of measure of a numeric property.
- Add “x-ogc-unit” in examples.
- Additional ones, like the URI of a definition should be added in an extension, if needed.
- Do not use “id” as the property name of an “id” property in the examples.
- Improve the wording in some normative statements.
- Consider adding a “primary-interval” role with a proper interval representation (array of two instants).

- Fix a typo (“roadSegement” instead of “roadSegment”).
- Remove broken dependency in Clause 13.
- The data types of queryables should not be restricted. What is supported will depend on the filter language. Add a note that filter languages may not support data types.

A pull request was created during the sprint. The pull request was discussed, approved, and merged in the Features API SWG meeting after the sprint with the following changes.

- Changed “x-ogc-property-index” to “x-ogc-property-seq.”
- For the languages to specify units of measurement, a keyword “x-ogc-unit-lang” was added with the only value “UCUM” (the default). A note was added to clarify that communities or other standards can add more languages.

During the code sprint, ldproxy was updated to support the changes, notably the new “x-ogc-unit” keyword:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://demo.ldproxy.net/strassen/collections/abschnitteaeste/
schema",
  "title": "Abschnitte und Äste",
  "description": "Als Abschnitt wird ein gerichteter Teil des Straßennetzes
bezeichnet, der zwischen zwei aufeinander folgenden Netzknoten liegt. Er
wird durch die in den Netzknoten festgelegten Nullpunkte begrenzt.<br>Als Ast
wird der Teil des Straßennetzes bezeichnet, der die Abschnitte untereinander
verkehrlich verknüpft und deshalb Teil des Netzknotens ist. Er wird durch
die im Netzknoten festgelegten Nullpunkte begrenzt. Eine Festlegung von Ästen
erfolgt nur, wenn sie Bestandteil des aufzunehmenden Straßennetzes sind.",
  "type": "object",
  "properties": {
    "kennung": {
      "title": "Kennung",
      "description": "16-stellige Kennung des Abschnittes oder Astes",
      "x-ogc-role": "id",
      "pattern": "^[0-9]{7}[A-Z][0-9]{7}[A-Z]$",
      "type": "string"
    },
    ...,
    "laenge_m": {
      "title": "Länge [m]",
      "description": "Länge des Abschnittes oder Astes (m)",
      "minimum": 0,
      "x-ogc-unit": "m",
      "type": "integer"
    },
    ...,
    "netzstand": {
      "title": "Datum des Netzstichtages",
      "x-ogc-role": "primary-instant",
      "format": "date-time",
      "type": "string"
    },
    "geometry": {
      "x-ogc-role": "primary-geometry",
      "format": "geometry-linestring"
```

```
      }
    }
}
```

4.1.4. OGC API — Maps

Sprint participants from UAB-CREAF and Ecere that serve on the OGC API — Maps SWG as editors of the OGC API — Maps — Part 1: Core candidate standard worked on clarifying the candidate standard and also added an informative annex for examples. In addition, example requests were provided for an area of London, using Ordnance Survey OpenMapLocal and Copernicus Sentinel-2 data to include in the OGC API — Maps Example annex.

The Sprint participants also compared results of the GNOSIS Map Server vs. the CREAF MiraMon server including assessing recent changes for Subsetting and Scaling requirements classes and support for the scale-denominator, center, and mm-per-pixel query parameters.

4.1.5. STAplus SensorThings API Extension

4.1.5.1. Statement of the Challenge

The STAplus use case for the code sprint is the following: A bus company “X-Bus” operates diesel buses on different routes in the city. X-Bus has ordered some electric buses to replace diesel buses. But, because there are not enough electric buses to replace the diesel buses on all routes, X-Bus would like to replace those routes first where the air quality is the lowest. X-Bus heard that a parcel delivery service “Green Delivery” captures air quality in the city based on sensors mounted on their delivery vehicles. Green Delivery provides access to available observations via an implementation of the STAplus extension of the OGC SensorThings API Standard. Access Control is in place for the air quality data captured by Green Delivery that makes the observations readable for their users only. After coming to an agreement with X-Bus, Green Delivery allows X-Bus to read observations within the area of the city for the last 30 days.

The actual technical challenge is to protect access to the STAplus implementation. A STAplus implementation can be used to provide multiple users Create/Update/Delete/Read access to manage the users own air quality measurements. The business logic to be implemented has two aspects: Integrity and confidentiality of the uploaded observations; and the air quality measurements. As described in the OGC Best Practice for using SensorThings API with Citizen Science, the implementation of the concept of ownership is one possible solution towards integrity. The implementation of integrity can take place in the implementation itself, which allows the execution of a well-defined test harness.

Regarding the confidentiality of observations available via STAplus, there are many different use cases that would filter the requested observations differently. The following use cases are typical examples.

- A First Responder may have access to air quality data near the incident location to determine a hazardous situation.

- Air quality data for a given area may only be readable to users in the area.
- Air quality data with extreme measurements may get blocked dynamically until validated.
- Access may be limited to display of observations for a given duration into the past, e.g., last 48 hours.
- Access may be limited over a given absolute time frame.
- A user may declare all of the user's data private which requires that all other users cannot access the observations of the user.

So, the challenging questions are:

- what would an architecture look like; and
- is it possible to implement a performant solution based on open standards?

4.1.5.2. Purpose

The work on the challenge shall prove that a solution can be based on open standards from OGC, IETF, and OpenID Connect. Further, the sprint participants sought to prove that a solution could be implemented without having to modify the existing OGC SensorThings API nor STAplus standards. To enforce spatio-temporal access control conditions, the sprint participants also sought to prove that the combined use with the OGC GeoXACML 3 standard provided a flexible interoperable solution.

4.1.5.3. Proposed Solution

The deployment of an OGC API implementation is described with OpenAPI providing the ability to enable authentication for endpoints of the deployed API. The available security schemes are listed on the OpenAPI Authentication page. Once the user's identity is known, different access control rules can be enforced.

The sprint participants proposed to enable OAuth2/OpenID Connect authentication on a codesprint demo endpoint which was based on the STAplus implementation by Secure Dimensions and the SensorThings API implementation by Fraunhofer.

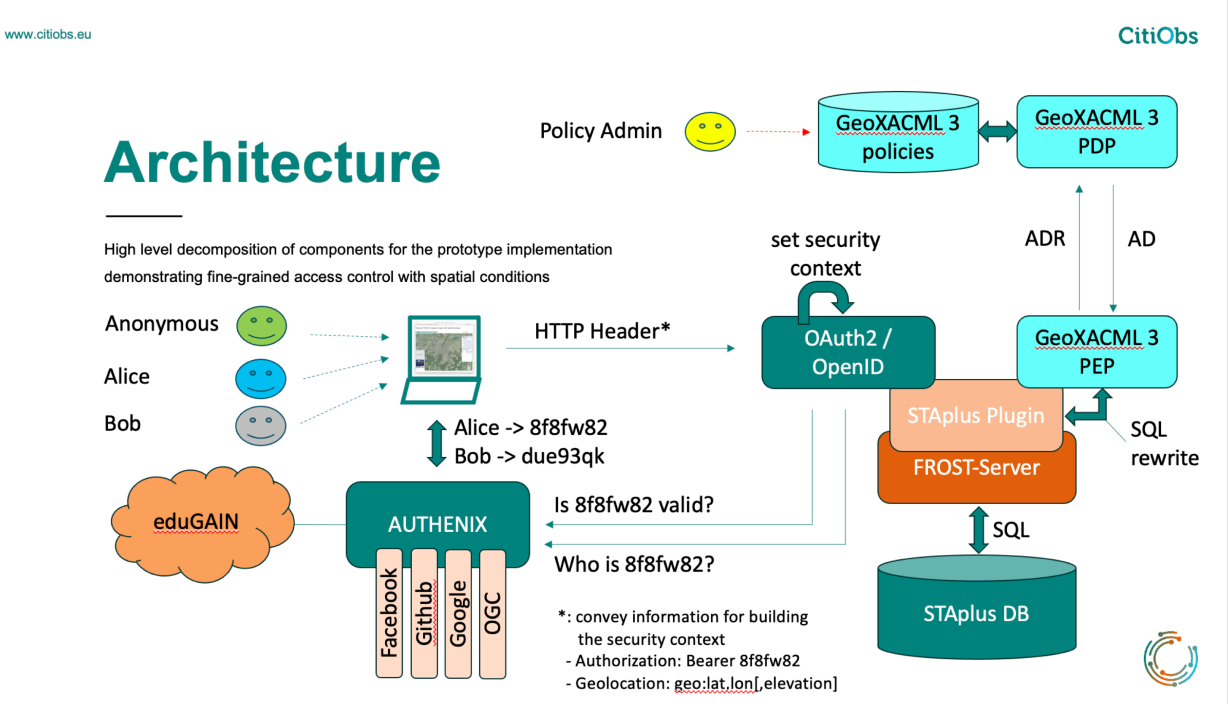


Figure 2 — The STAplus and GeoXACML based architecture implemented by the CitiObs project

For the code sprint, the participants proposed extending the existing STAplus implementation with a GeoXACML Policy Enforcement Point (PEP) to enforce policies executed by the GeoXACML 3.0 Policy Decision Point (PDP). The participants planned to leverage the GeoXACML 3.0 implementation from Secure Dimensions and the XACML 3.0 implementation from Thales and also planned to visualize the access control results (enforcing a GeoXACML 3.0 policy) by using the generic STAplus visualization application TAPIS implemented by CREAF.

4.1.5.4. Achievement

During the Code Sprint, a demo was implemented that included the following components.

- STAplus endpoint: https://ogc.demo.secure-dimensions.de/staplus/v1.1
- TAPIS endpoint: https://www.tapis.grumets.cat/

As per the architecture diagram, the (Geo)XACML policy controls the SQL query rewriting. To demonstrate the functionality, a GeoXACML 3.0 wrapper was implemented. To leverage this functionality, a policy must be written using XACML 3 obligations. An obligation can be understood as an object that gets executed by the PEP. In this implementation, the obligation targets to rewrite the SQL query that was constructed from the HTTP request. To ease the policy construction, the sprint participants used an extension to ALFA, a JavaScript alike notation for writing policies which resolves to the following simplistic structure of a policy:

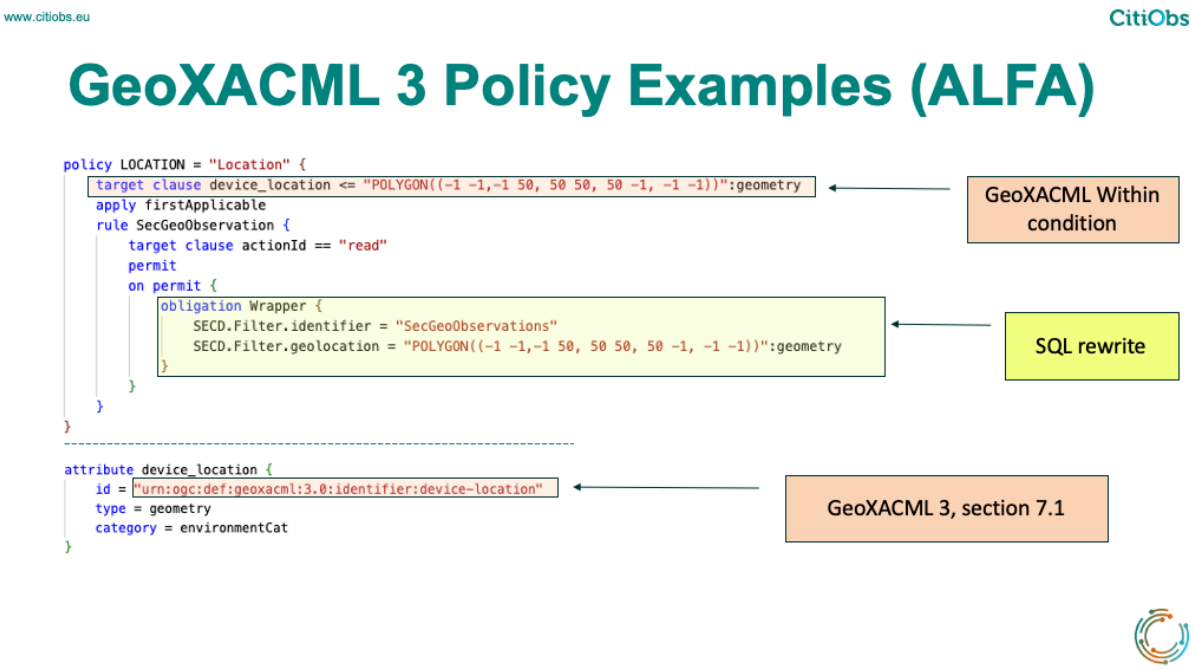


Figure 3 — Fragment from example GeoXACML policy

The ”interesting” part of the policy is the obligation that undertakes the SQL rewrite. The implementation is generic, as it supports declaring obligations based on parameterized SQL expressions. In technical terms, the SQL expression must leverage the actual database structure but also honor the relations among the STAplus entities.

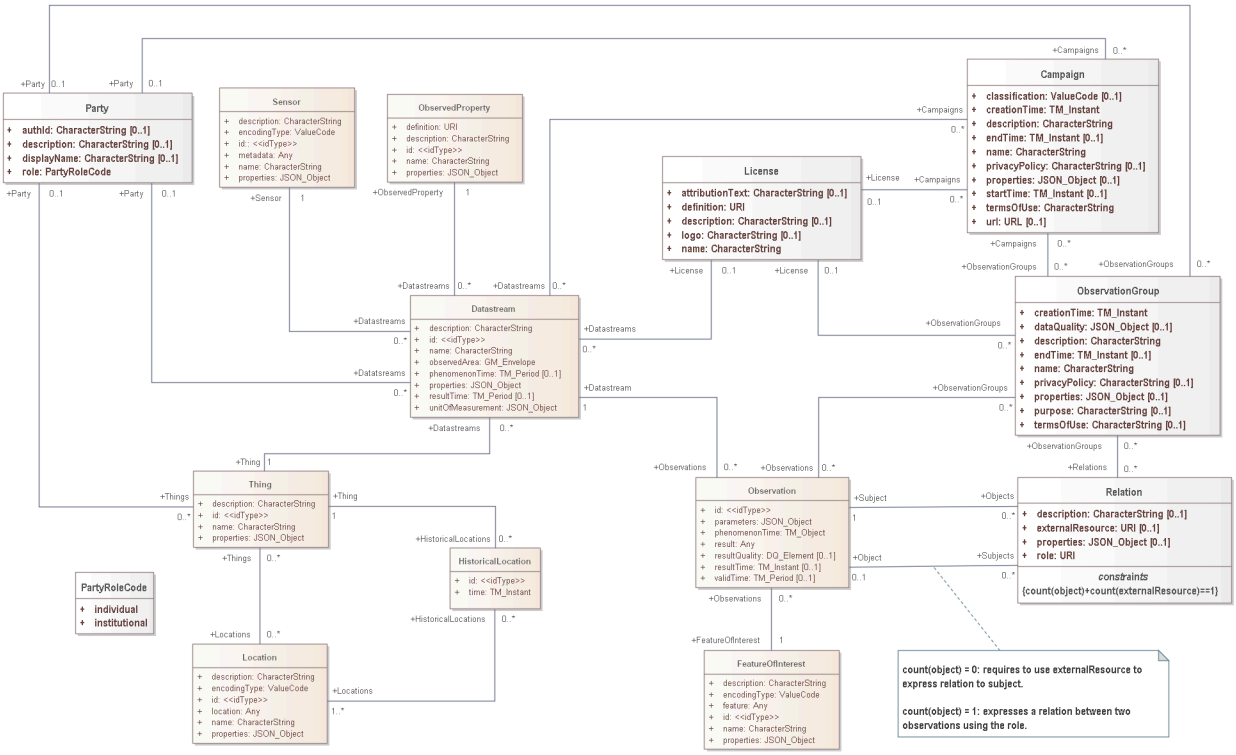


Figure 4 — STAplus Entity Type diagram

The following example illustrates the obligation to filter on “my” obligations.

```
{
  "entries": [
    {
      "tableName": "OBSERVATIONS",
      "wrappers": [
        {
          "@class": "de.fraunhofer.iosb.ilt.frostserver.persistence.pgjooq.
utils.validator.SecurityWrapperMulti",
          "wrappers": [
            {
              "condition": {
                "@class": "de.fraunhofer.iosb.ilt.frostserver.persistence.
pgjooq.utils.validator.CheckFixed",
                "valid": true
              },
              "wrapper": {
                "@class": "de.fraunhofer.iosb.ilt.frostserver.persistence.
pgjooq.utils.validator.SecurityWrapperJoin",
                "joins": [
                  {
                    "targetTable": "DATASTREAMS",
                    "targetAlias": "ds",
                    "joinOnSql": "t.\"DATASTREAM_ID\" \u003d ds.\"ID\"",
                    "joinType": "INNER"
                  },
                  {
                    "targetTable": "PARTIES",
                    "targetAlias": "pa",
                    "joinOnSql": "ds.\"PARTY_ID\" \u003d pa.\"ID\"",
                    "joinType": "LEFT"
```

```
                }
              ],
              "where": "pa.\"AUTHID\" \u003d ?",
              "usernameParameter": true
            }
          }
        ]
      }
    ]
  }
}
```

Wrapper to filter observations

A wrapper was implemented to filter observations according to the policy. The SQL wrapper applies to the table “OBSERVATIONS” and uses a join on “DATASTREAMS” and “PARTIES” with a final WHERE clause that conditions the Party/authId eq username. The joins follow the relations per the STAplus UML diagram. But instead of using the entity type names from the UML diagram, the actual table names are used.

The visualization of the “my observations” was done via the TAPIS implementation. First, a user needs to select the Party representing the user. Then the system connects to the Datastreams for the Party. Once a Datastream is selected, the user can query the observations.

Case 1: Anonymous access

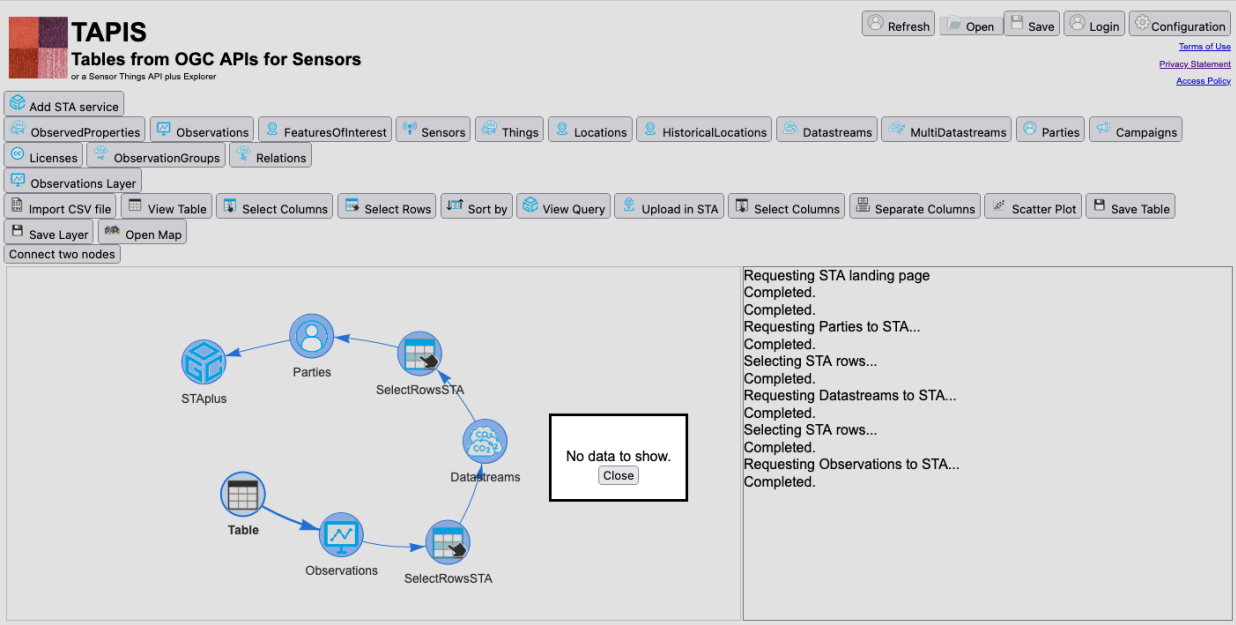


Figure 5 — Screenshot of the interface when a user is NOT logged into the TAPIS system

Figure 5 shows that for anonymous access, no observations are returned.

Case 2: User Long John Silver logged in access

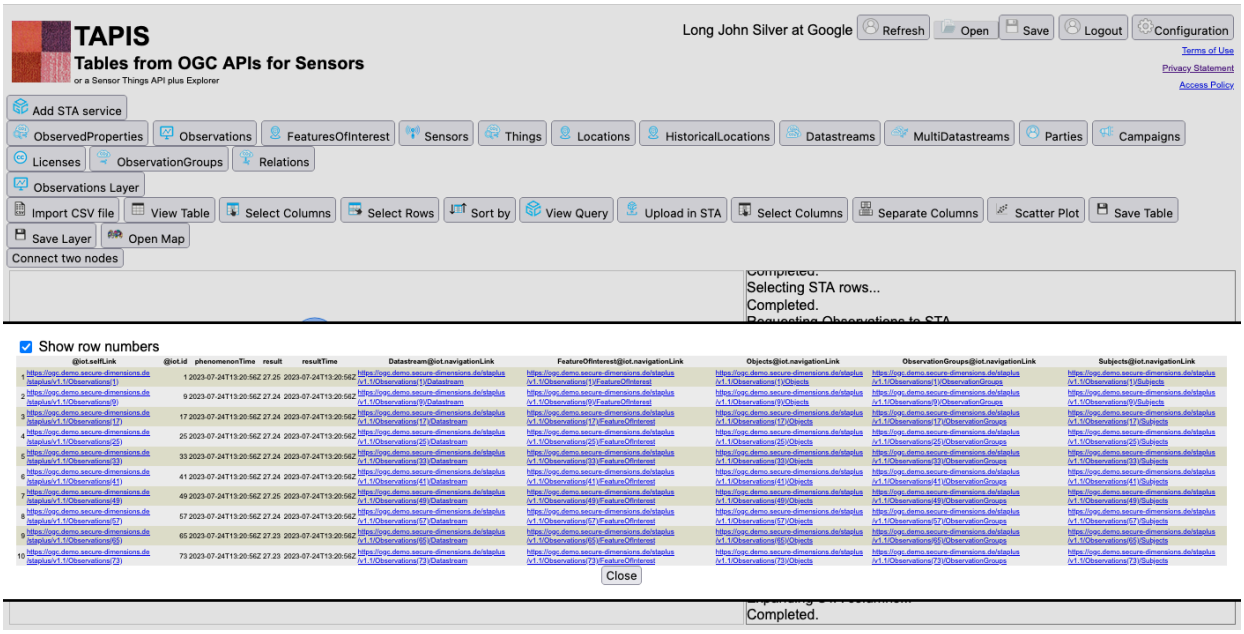


Figure 6 — Screenshot of the interface showing observations when a user has been authenticated

Figure 6 shows that Long John Silver can fetch his observations.

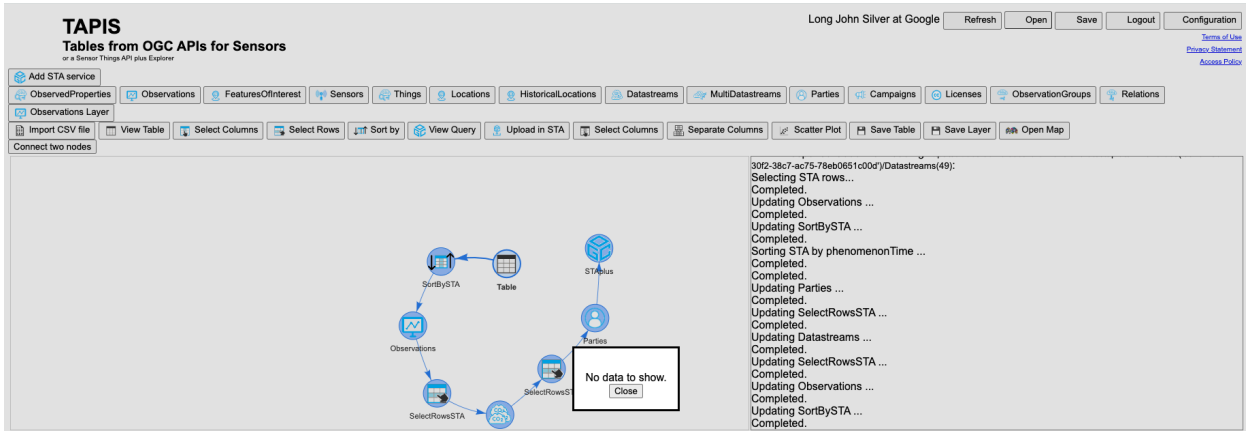


Figure 7 — Screenshot of the interface when authenticated user is prevented from accessing other user's observations

Figure 7 shows that Long John Silver cannot fetch Joan's observations; the attempt to access observations results in "no data".

4.2. Approved and Candidate OGC Encoding Standards

4.2.1. OGC Features and Geometries JSON

The JSON-FG candidate Standard was approved by the OGC Architecture Board for Public Comment shortly before the code sprint. The JSON-FG team held one breakout session during the sprint to answer questions on the specification and provide an overview over changes introduced in the new draft version 0.2.

The following list summarizes JSON-FG-related results achieved during the sprint.

- Participants from Geonovum tested the new JSON-FG driver in GDAL, see Clause 4.3.6 for details.
- A free and open source json-fg-validator tool was created as an executable test suite to test JSON FG documents against the requirements. The initial implementation provides a framework to test and report. The first pass covered most of the “Core” Requirements Class.

  The tool is implemented in Python and contributions are welcome and encouraged.

```
(opengeospatial) tomkralidis@aruba:~/Dev/opengeospatial/json-fg-validator$ json-fg-validator ets validate  toronto-city-hall.json
Opening toronto-city-hall.json
Validating toronto-city-hall.json
Detected JSON FG
{
    "ets-report": {
        "summary": {
            "PASSED": 4,
            "FAILED": 0,
            "SKIPPED": 4
        },
        "tests": [
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/metadata",
                "code": "PASSED"
            },
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/coordinate-dimension",
                "code": "SKIPPED",
                "message": "Geometry is null"
            },
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/geometry-wgs84",
                "code": "SKIPPED",
                "message": "Geometry is null"
            },
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/instant",
                "code": "PASSED",
                "message": "Passes given data is compliant/valid to schema"
            },
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/instant-and-interval",
                "code": "SKIPPED",
                "message": "Skipping given time is null"
            },
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/interval",
                "code": "PASSED",
                "message": "Passes given data is compliant/valid to schema"
            },
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/utc",
                "code": "SKIPPED",
                "message": "Time is null"
            },
            {
                "id": "http://www.opengis.net/spec/json-fg-1/0.2/req/core/schema-valid",
                "code": "PASSED"
            }
        ]
    }
}
```

Figure 8 — Screenshot of the json-fg-validator tool providing an ETS report.

- Development of a json-fg-linter tool was started by Geonovum. The linter rules are based on JSON-FG requirements classes Core, 3D and Feature Types, and Schemas. In addition, the linter does some consistency checks that are useful but not based on explicit requirements.
- An initial version of a json-fg-java library was developed by Geonovum. The library aims to make encoding and decoding of JSON-FG documents as easy as possible in Java. The status of the library is experimental. The implementation is compatible with v0.2.1 of the JSON-FG specification.
- A JSON-FG map client was deployed by sprint participants that supported viewing JSON-FG documents by drag-and-drop.
- MUDDI-conformant data was loaded into MariaDB and published via an OGC Web API, including in JSON-FG; see Clause 4.3.1 for details.
- The ldproxy implementation was updated to JSON-FG version 0.2.1 during the sprint, see Clause 4.3.2 for details.

One topic during the breakout was support for different coordinate reference systems and the GeoJSON compatibility. Since the sprint took place in London, the British National Grid (EPSG:27700) was used as the coordinate reference system in example requests to an API implementing OGC API — Features as follows.

- The MariaDB results section includes a “Manhole” feature being presented as JSON-FG using the EPSG:27700 CRS as provided from the MariaDB feature server.
- An airport feature from the OS Open Zoomstack dataset in the default coordinate reference system (OGC:CRS84):


```
{
  "type": "FeatureCollection",
  "numberReturned": 1,
  "timeStamp": "2023-11-08T06:30:45Z",
  "featureType": "Airport",
  "featureSchema": "https://demo.ldproxy.net/zoomstack/collections/
airports/schema",
  "coordRefSys": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
  "conformsTo": [
    "http://www.opengis.net/spec/json-fg-1/0.2/conf/core",
    "http://www.opengis.net/spec/json-fg-1/0.2/conf/types-schemas"
  ],
  "geometryDimension": 0,
  "features": [
    {
      "type": "Feature",
      "id": 1,
      "geometry": {
        "type": "Point",
        "coordinates": [
          -1.2918826,
          59.8783475
        ]
      },
      "properties": {
        "name": "Sumburgh Airport"
      },
      "place": null,
      "time": null
    }
  ]
}
```


- The same airport in the British National Grid (EPSG:27700):


```
{
  "type": "FeatureCollection",
  "numberReturned": 1,
  "timeStamp": "2023-11-08T06:32:22Z",
  "featureType": "Airport",
  "featureSchema": "https://demo.ldproxy.net/zoomstack/collections/
airports/schema",
  "coordRefSys": "http://www.opengis.net/def/crs/EPSG/0/27700",
  "conformsTo": [
    "http://www.opengis.net/spec/json-fg-1/0.2/conf/core",
    "http://www.opengis.net/spec/json-fg-1/0.2/conf/types-schemas"
  ],
  "geometryDimension": 0,
  "features": [
    {
```

```
      "type": "Feature",
      "id": 1,
      "geometry": null,
      "properties": {
        "name": "Sumburgh Airport"
      },
      "place": {
        "type": "Point",
        "coordinates": [
          439742.86,
          1110569.17
        ]
      },
      "time": null
    }
  ]
}
```


- The same airport in the British National Grid (EPSG:27700), but for GeoJSON compatibility also with the geometry in OGC:CRS84:


```
{
  "type": "FeatureCollection",
  "numberReturned": 1,
  "timeStamp": "2023-11-08T06:32:41Z",
  "featureType": "Airport",
  "featureSchema": "https://demo.ldproxy.net/zoomstack/collections/
airports/schema",
  "coordRefSys": "http://www.opengis.net/def/crs/EPSG/0/27700",
  "conformsTo": [
    "http://www.opengis.net/spec/json-fg-1/0.2/conf/core",
    "http://www.opengis.net/spec/json-fg-1/0.2/conf/types-schemas"
  ],
  "geometryDimension": 0,
  "features": [
    {
      "type": "Feature",
      "id": 1,
      "geometry": {
        "type": "Point",
        "coordinates": [
          -1.2918826,
          59.8783475
        ]
      },
      "properties": {
        "name": "Sumburgh Airport"
      },
      "place": {
        "type": "Point",
        "coordinates": [
          439742.86,
          1110569.17
        ]
      },
      "time": null
    }
  ]
}
```

4.2.2. OGC CDB

The Sprint participants from Ecere Corporation and FlightSafety International Visual Systems collaborated on work to advance the development of CDB 2.0. These participants set up a new draft CDB 2.0 GeoPackage Data Store candidate standard with initial content. The candidate standard defines a data store implementing the CDB 2.0 Core Conceptual Model based on the OGC GeoPackage data container and GeoPackage extensions for storing different data types.

The candidate standard defines a conceptual model and data stores for encoding different types of data, including detailed 3D environments, according to that conceptual model with a focus on deterministic performance and scalability. This version of CDB is envisaged to enable a variety of use cases ranging from mobile computing at the edge to large scale data repositories, including training, simulation, digital twins, and acting as an efficient backing data store for OGC API implementations.

4.2.3. GML and JSON-FG schemas for the MUDDI Core Logical Model

To support sprint participants with encoding MUDDI-conformant data, interactive instruments GmbH derived GML and JSON-FG application schemas from the Core Logical Model of MUDDI.

ShapeChange is a free and open source tool written in Java that takes application schemas constructed according to ISO 19109 from a UML model and derives implementation representations. Implementation representations include GML application schemas, JSON Schemas, SQL DDL, feature catalogs, etc.

For the code sprint, a ShapeChange configuration for the Core Logical Model was created that derives a GML application schema, a JSON-FG schema, and an HTML feature catalog.

The results were made available in a folder of the GitHub repository used for the code sprint. In particular:

- GML application schema
- JSON-FG Schema

Participants from the MUDDI working group and interactive instruments discussed several modelling aspects of the Core Logical Model before deriving the results. For the sprint, the following decisions/changes were made to the Core Logical Model.

- Some errors in the model were fixed (e.g., undefined data types like “int” instead of “Integer” or duplicate model elements with the same name).
- Geometry types from ISO 19107:2003 are used; geometry properties have a maximum multiplicity of “1”.
- The application schema Codelist Register — MUDDI Core should be parallel to the Core Logical Model, not nested into it.

The following are remarks on the generated schemas.

- The GML application schema was generated using the GML encoding rule from ISO 19136:2020. For simplicity, codelist values are represented as plain strings, with additional values having prefix “other: .” The target namespace, the namespace prefix, and XML Schema Definition (XSD) filename are placeholders.
- In the JSON-FG application schema, “$id” value is a placeholder. Schema references use JSON Pointers, not the anchors, because validator support for JSON Pointers is better compared to anchor references.

The schemas were used, for example, by participants from Safe Software, see Clause 4.3.4 for details.

4.2.4. MUDDI logical schema experimentation

Sprint participants from Ordnance Survey and the Geospatial Commission sought to examine how a schema that conformed to MUDDI could be created to support the ingestion of typical utility source data. The participants organized their code sprint work into creation of a backend to store MUDDI-compliant data and a front end to display the data. To facilitate the task, open utility data was transformed into MUDDI-compliant data.

The Geospatial Commission and Ordnance Survey actively supported the MUDDI logical schema experimentation because the UK’s National Underground Asset Register (NUAR) data model is an implementation of the MUDDI model [21]. The Geospatial Commission therefore provided a range of artifacts and sample data in support of the experimentation.

The sprint participants started by reviewing the MUDDI extended conceptual model. All of the feature types in the model were translated associated tables in a database, as illustrated in Figure 9. The sprint participants also considered the inheritance between feature types, creating tables for ‘real’ feature types which inherit attribution from ‘abstract’ feature types in the conceptual model.

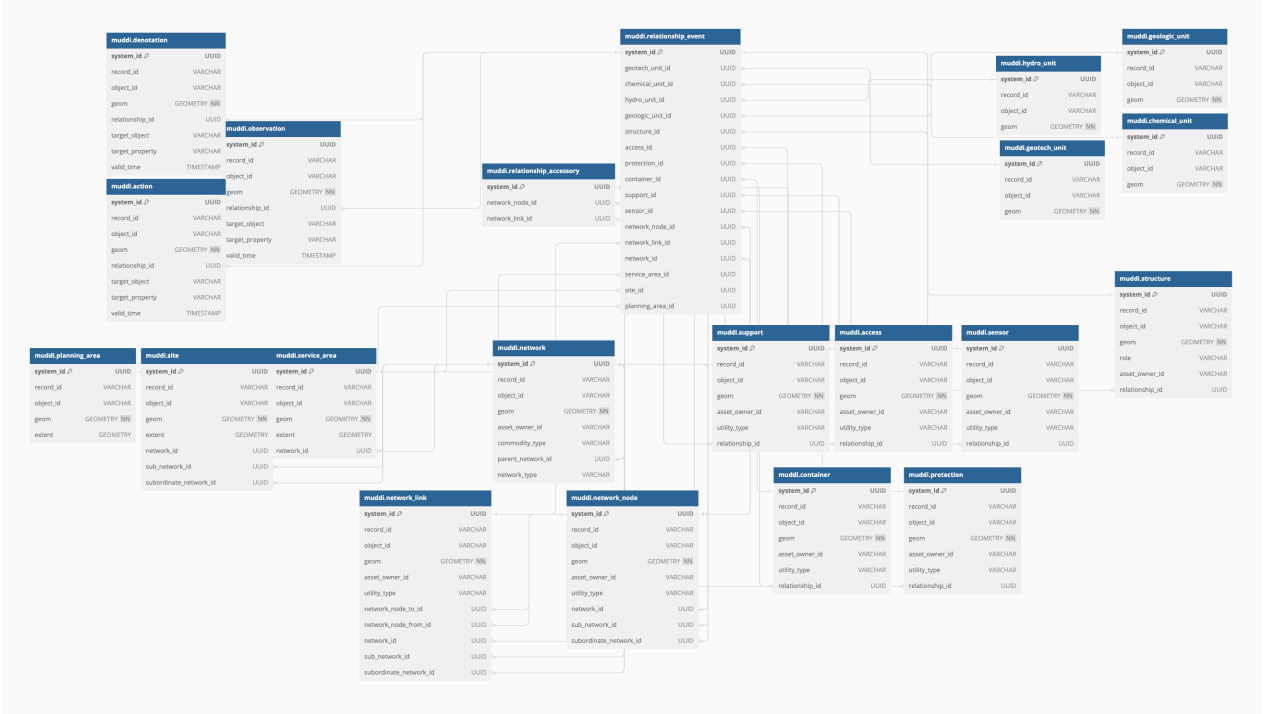


Figure 9 — Illustration of a MUDDI logical schema for a relational database

The figure shows the tables that were created and how the tables relate to one another. The figure consists of tables for each real feature type. These tables are mostly linked by foreign keys, e.g., for linking conveyance information against network information.

Some fields are related to multiple tables, so in order to maintain the relationship structure, the sprint participants had to create relationship tables to document relationships between different tables. As shown on the figure, there is a 'big' table in the middle which ensures that a user can record the relationships between a MUDDI Event type and any MUDDI Object type. Another example of a relationship table that had to be created to support representation of MUDDI in a relational model is the accessory table, which links Network Accessories (e.g., containers and sensors) to the relevant network conveyance (node or link). The figure has been created using an online tool called dbdocs.io.

One thing that the sprint participants found unclear in the candidate standard was the relationship between a network, subnetworks, and subordinate networks. The relationships between these types of networks is illustrated in Figure 10. A network is the entirety of the graph consisting of all nodes and edges, as highlighted by the blue oval on the diagram. The subnetwork is a geographic subset of the network, for example one owned by a county and is illustrated on the diagram through the three green ovals. There is also a subordinate network which is based on an asset property rather than geographic subsetting.

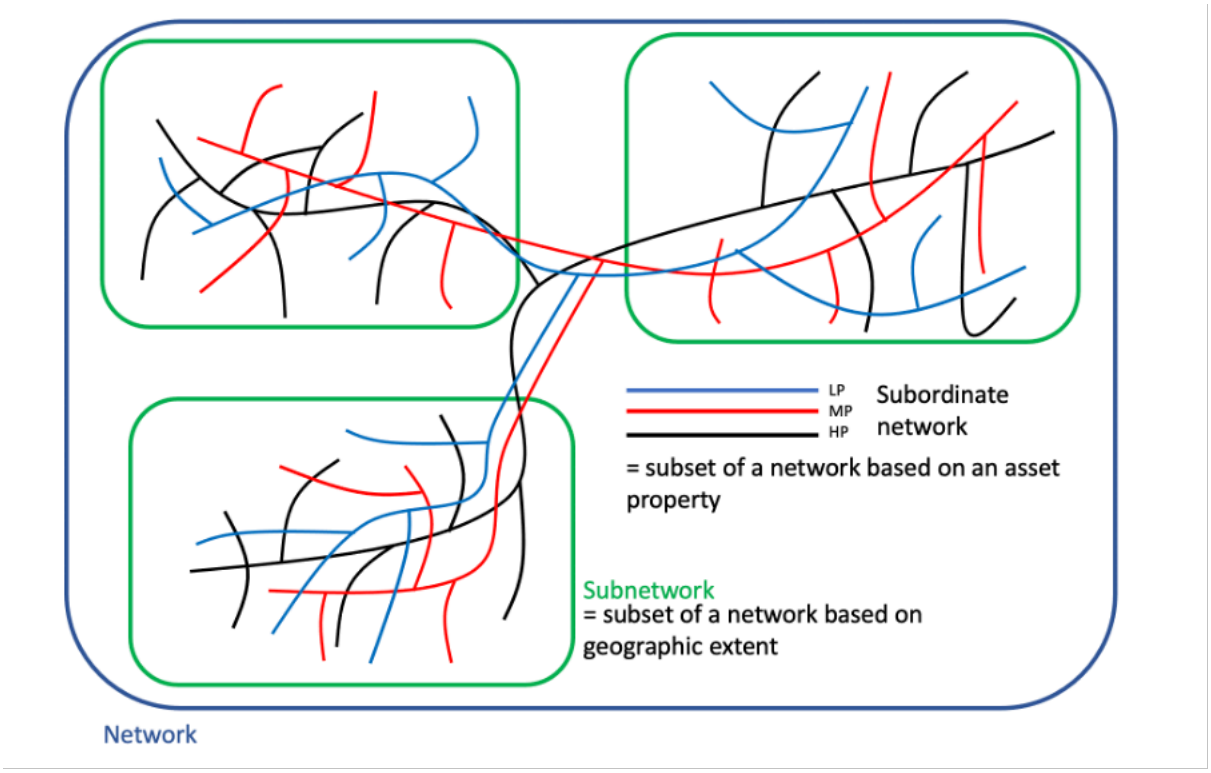


Figure 10 — Relationships between types of networks in MUDDI

To demonstrate that sample data conforming to the logical schema could be displayed on a web application, the sprint participants uploaded the data into a pygeoapi instance. The pygeoapi instance was then linked into a web application. A screenshot of the web application is shown in Figure 11.

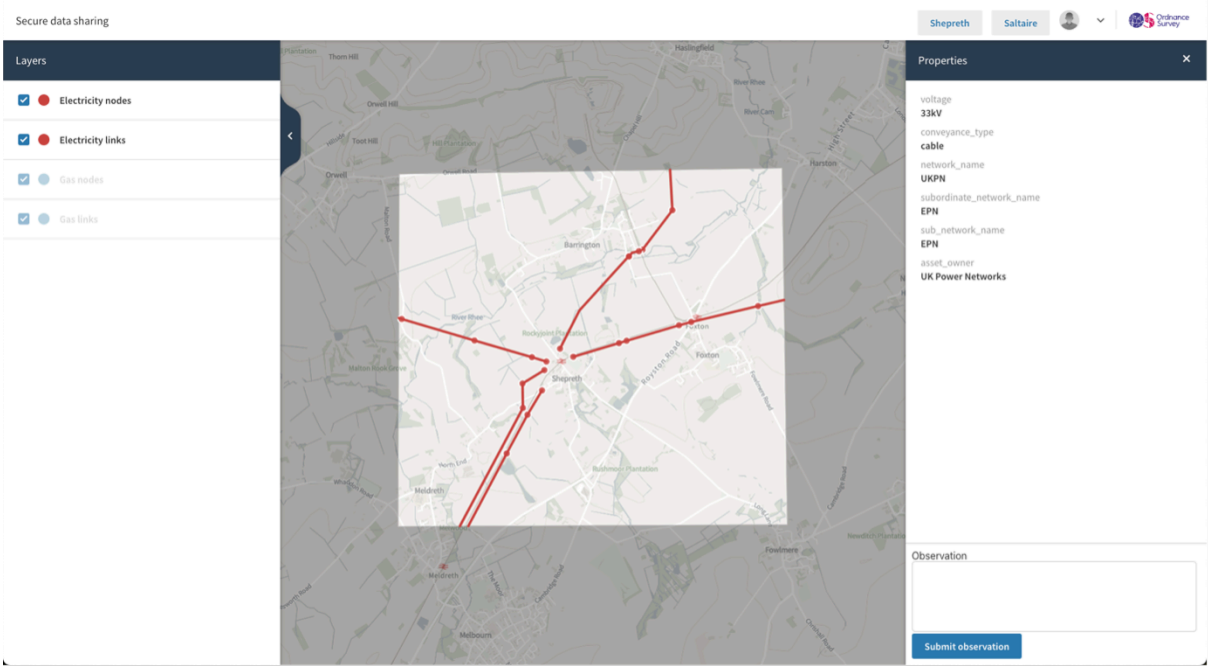


Figure 11 — A screenshot of a web application displaying underground asset data

Future work on the storage aspects of the MUDDI task could involve using the MUDDI logical schema as a template for MUDDI compliance. The logical schema does not currently have much attribution about the assets themselves. Therefore any implementation would be expected to introduce additional attributes. The code for the logical schema is available at: https://github.com/Geovation/muddi_compliance

4.2.5. Analysis of MUDDI by IIT Bombay

Sprint participants from IIT Bombay conducted an analysis of the MUDDI model, in relation to the physical data that had been supplied for the code sprint to determine the potential use of the logical model (represented in the schemas) to support harmonization of physical data which had been synthesized to represent data from different utility providers.

Several datasets had been provided, therefore half of the first day of the code sprint was spent reviewing the datasets and developing an understanding of the content that had been provided. One of the observations made during the review of the datasets was that some of the data was in different languages (e.g. English, German, etc.). A screenshot of the data displayed on FME Data Inspector is shown in Figure 12.

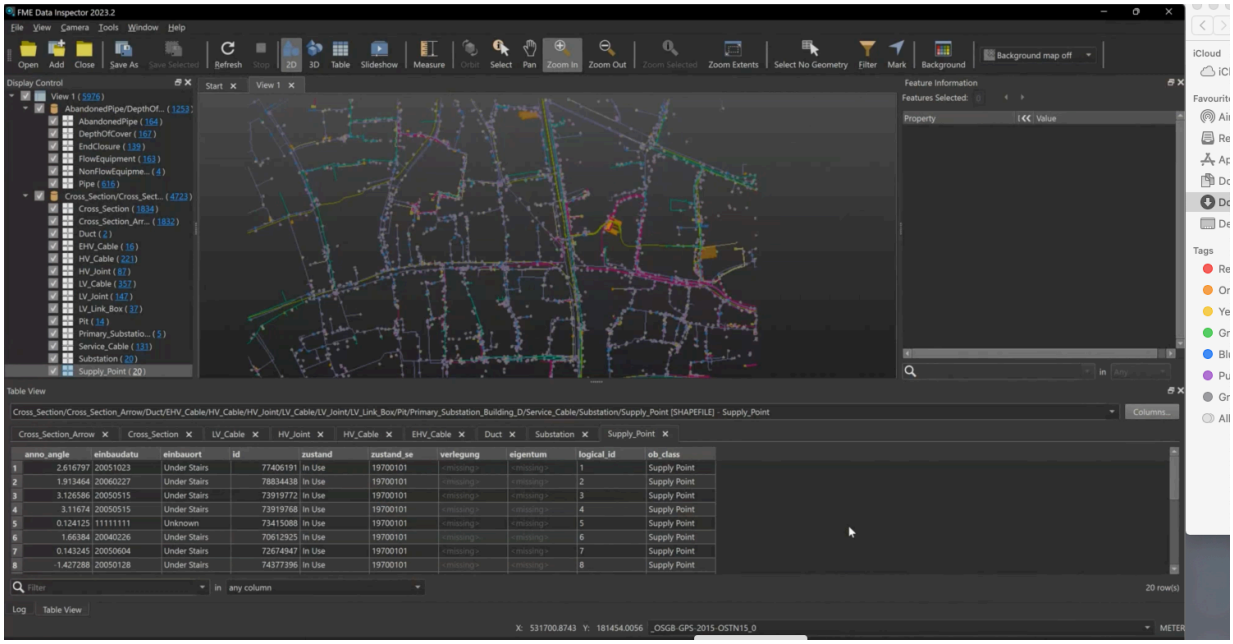


Figure 12 — Screenshot of MUDDI data displayed in FME Data Inspector

Another observation made during the review of the datasets was that some of the data was three-dimensional (3D). Part of the challenge therefore was whether the sprint participants could identify commonality between the MUDDI schemas and those of the synthetic datasets that had been supplied to represent data from gas and electricity providers.

Once the data had been converted to JSON, a similarity comparison was conducted. Figure 13 shows such a similarity comparison across synthetic data representing data from gas providers.

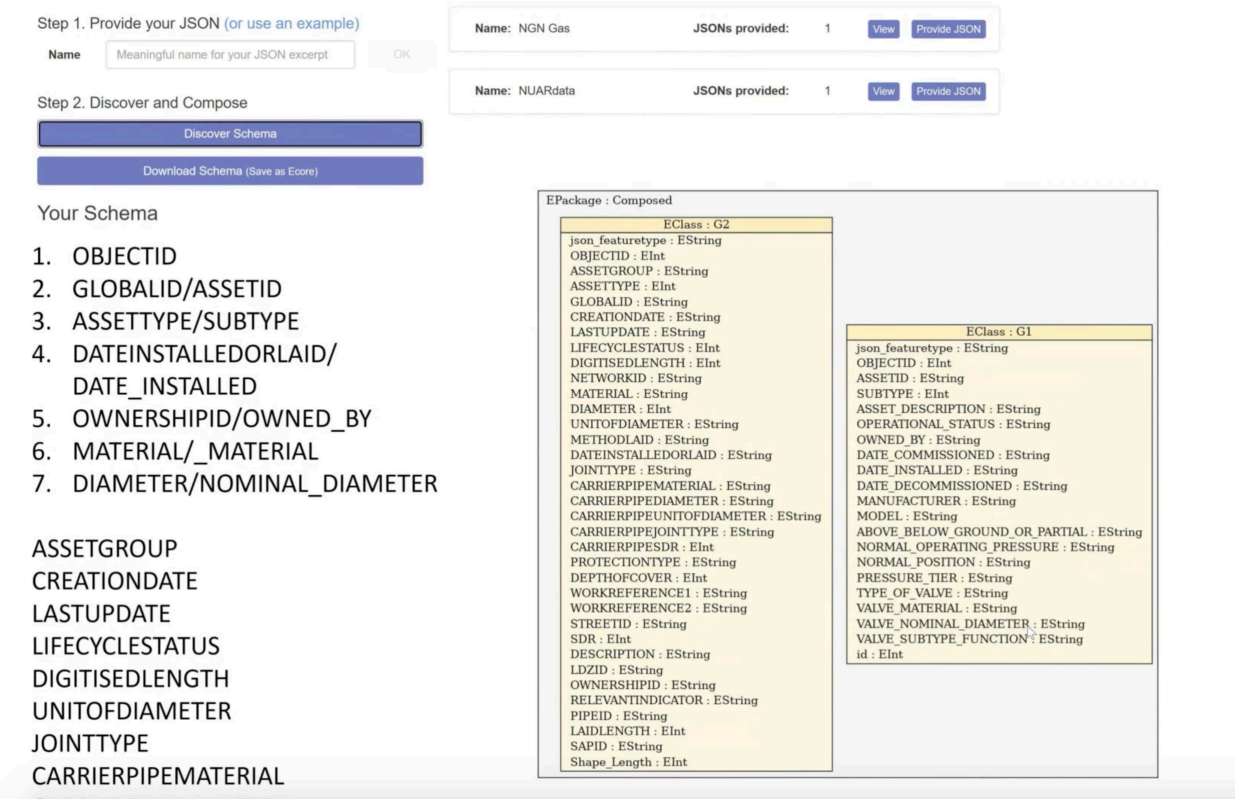


Figure 13 — Comparison of a sample of feature properties from gas provider data

A similar comparison was conducted between feature properties of synthetic data representing data from electricity providers. This comparison is illustrated in Figure 14.

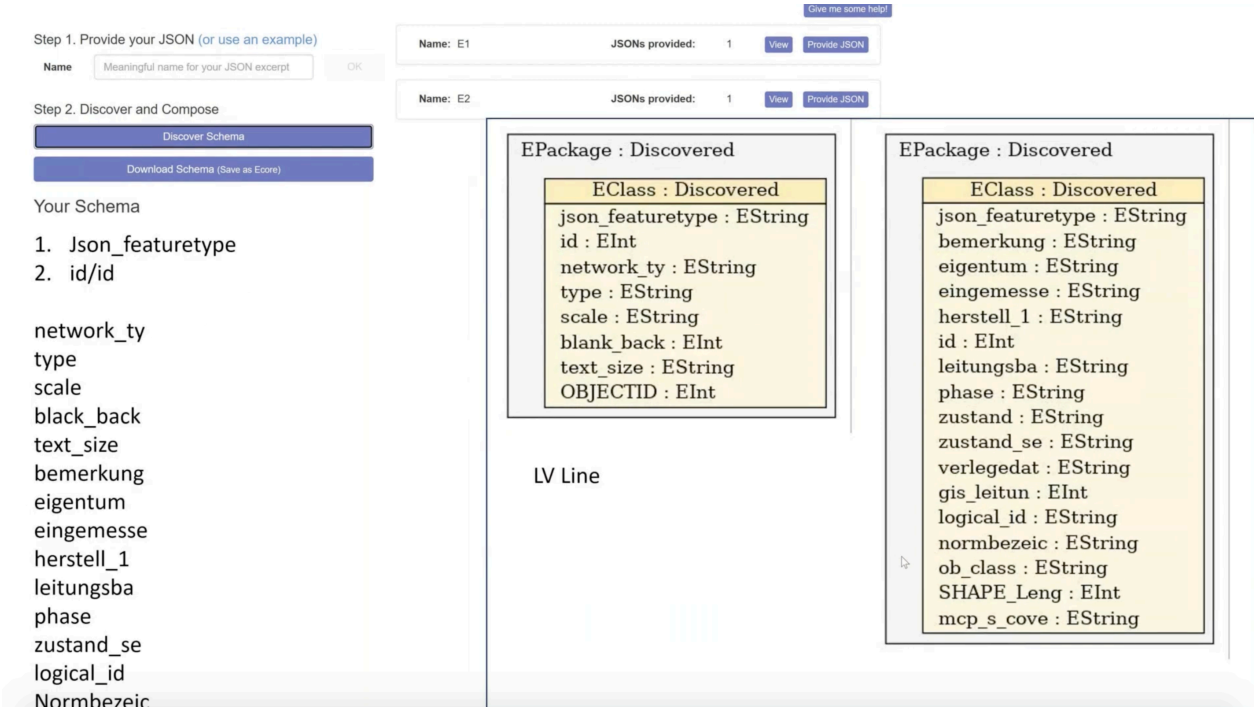


Figure 14 — Comparison of a sample of feature properties from electricity provider data

Sprint participants from IIT Bombay then went on to review other anonymized and synthetic datasets, and during that review it was noticed that across the datasets there were three electric provider datasets that had some level of commonality. The sprint participants therefore proceeded to use that insight to import the data into a PostgreSQL/PostGIS database and then to display it in QGIS, as illustrated in Figure 15.

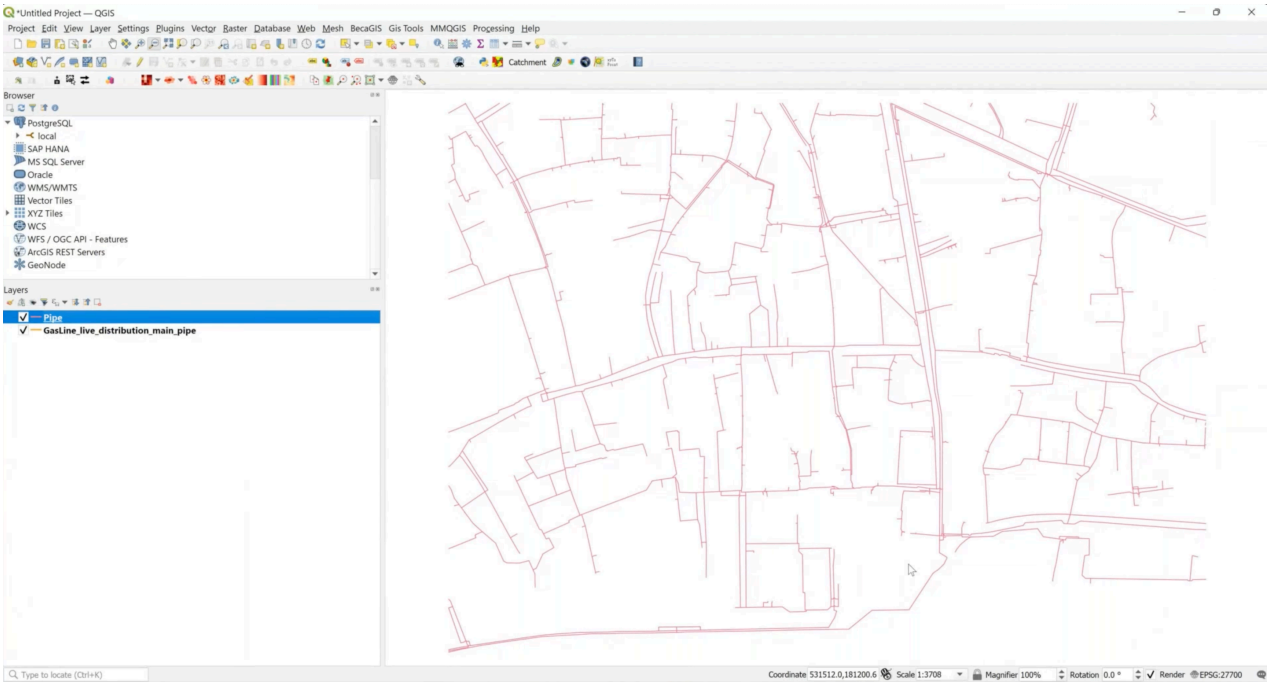


Figure 15 — A sample of MUDDI-harmonized data uploaded to a PostGIS database and displayed on QGIS

4.3. Products applied to OGC Standards in this code sprint

4.3.1. MariaDB

4.3.1.1. Summary of code sprint activities

Sprint participants from MariaDB worked on the following items.

- Attended and participated in the following breakout sessions:
  - MUDDI Breakout Sessions
  - JSON-FG Breakout session
  - OGC API — Records Breakout session
  - OGC API — Processes Breakout session
  - OGC API — Features Breakout session

- Loaded London MUDDI data into MariaDB.
- Updated the MariaDB OGC API — Records implementation to align with the current state of the Part 1 editor's draft.
- Updated the MariaDB OGC API — Processes implementation to align with the current state of the Part 1 & 2 editor's drafts.

4.3.1.2. Code sprint results

4.3.1.2.1. MUDDI

The goal of MariaDB's MUDDI work was to load MUDDI data into MariaDB and then serve the data out through OGC APIs. MUDDI data was loaded into MariaDB and could be accessed through OGC APIs at the following landing page:

- https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv?instance=London_2023.

The following image of MariaDB's Geospatial GeoPortlet illustrates the MUDDI data layers loaded into MariaDB.

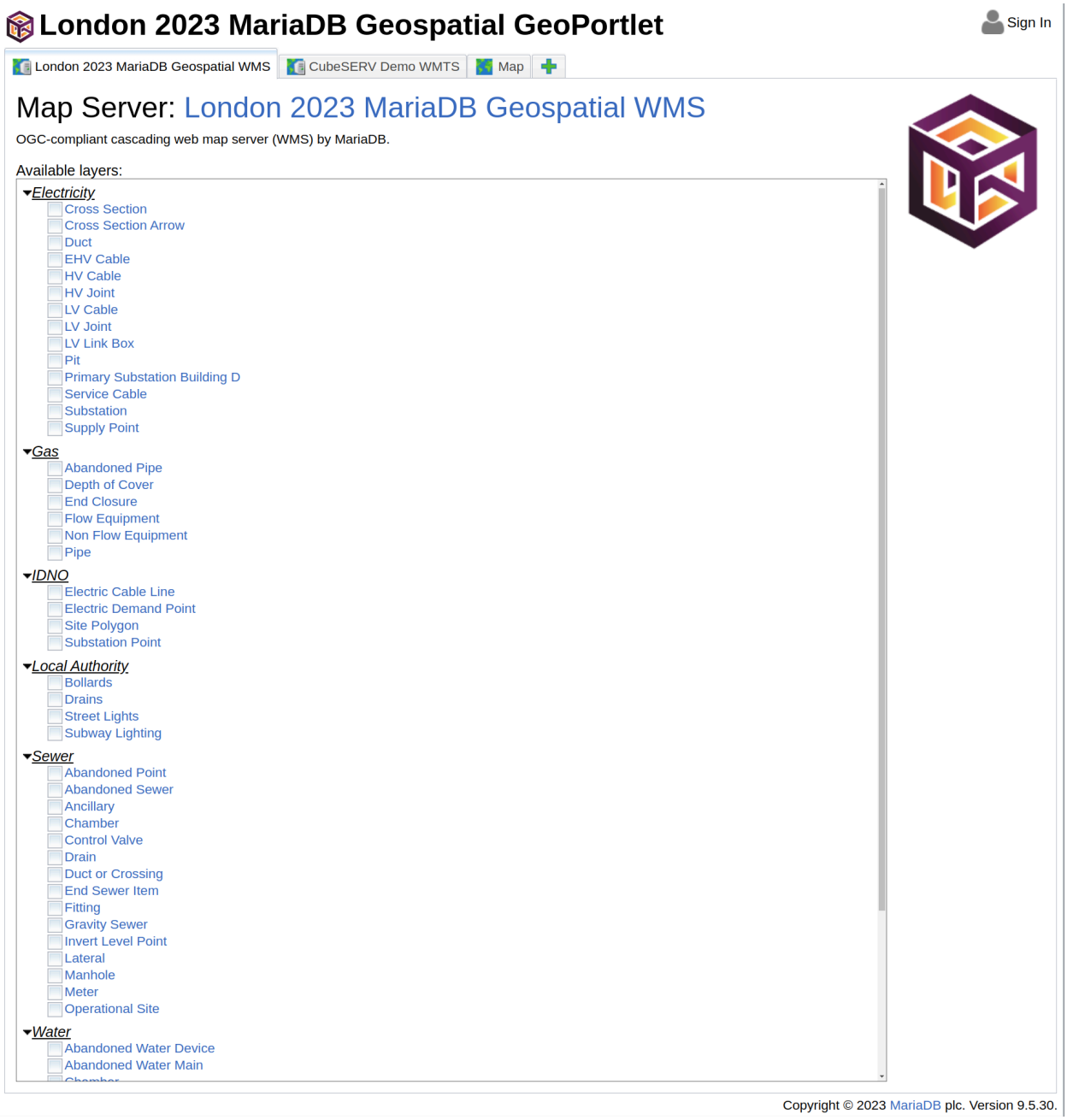


Figure 16 — MariaDB's Geospatial GeoPortlet listing layers created from MUDDI conformant data

The following image illustrates several of these layers, "Distribution Main," "Hydrant," and "Manhole" being accessed through the Maps API:

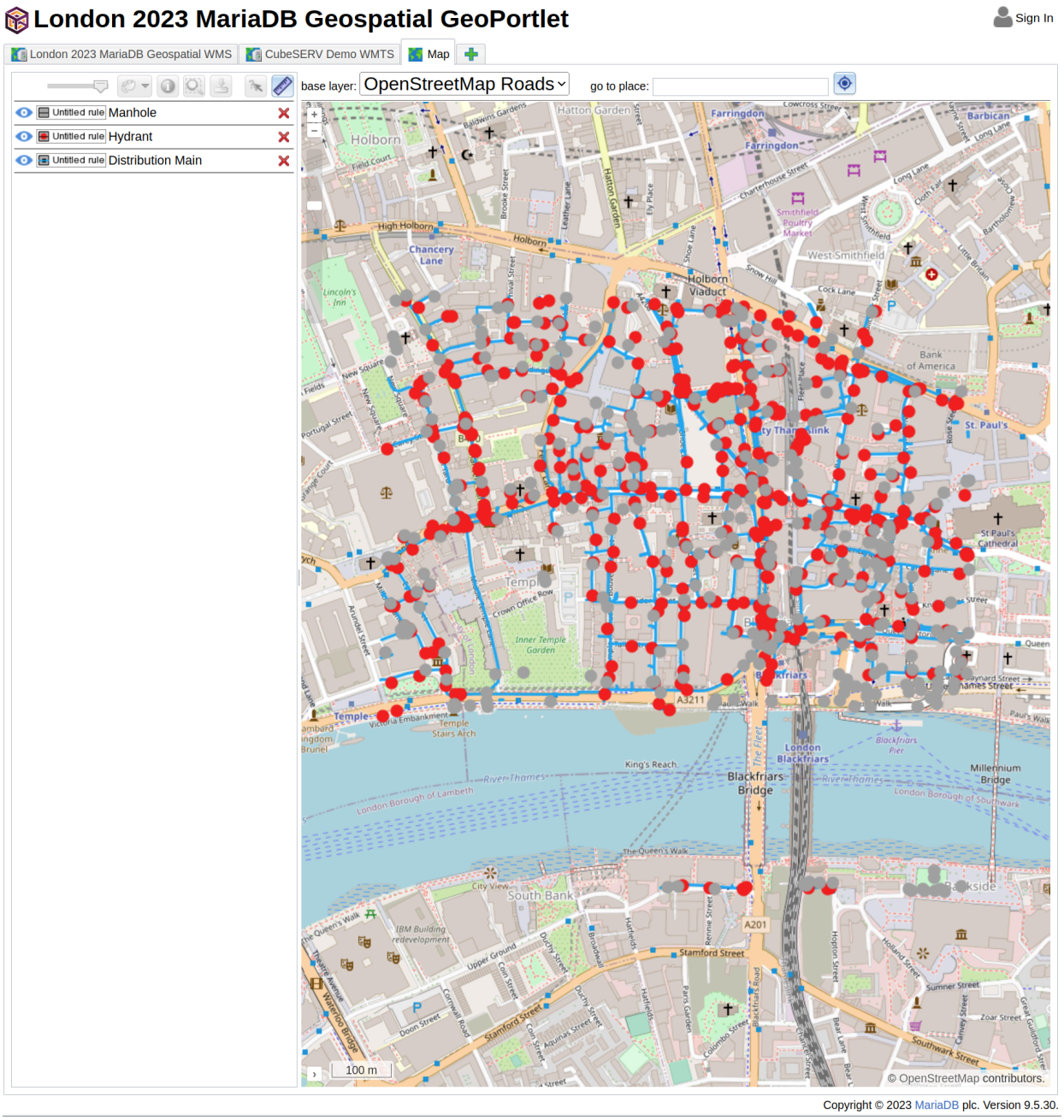


Figure 17 — Layers displayed on a map on MariaDB's Geospatial GeoPortlet

The following image illustrates one of these layers, "Manhole," being accessed using the Features API:

Manhole

[sign in]

Collection ID: Manhole

show context layer

WGS 84 Geographic Extent:

Minimum Latitude: 51.50712395675871
Minimum Longitude: -0.1151666277311117
Maximum Latitude: 51.51815088288766
Maximum Longitude: -0.09754299120417219

Schema

The vector features in this collection have the following schema:

Column Name	Column Type
GEOMETRY	point geometry
ANCILLARYR	integer (max length: 9)
ENABLED	integer (max length: 4)
DATECREATE	date
DATEMODIFI	date
GISID	integer (max length: 9)
WATERTRACE	integer (max length: 9)
REFERENCE	string (max length: 12)
ALTERNATER	string (max length: 12)

(expand to show entire schema)

Coordinate Reference Systems

The native coordinate reference system of this collection is:
OSGB36 / British National Grid (http://www.opengis.net/def/crs/EPSG/0/27700)

The following other coordinate reference systems are also available:
WGS 84 (http://www.opengis.net/def/crs/EPSG/0/4326)
WGS 84 (CRS84) (http://www.opengis.net/def/crs/OGC/1.3/CRS84)
WGS 84 / Pseudo-Mercator (http://www.opengis.net/def/crs/EPSG/0/3857)
WGS 84 / World Mercator (http://www.opengis.net/def/crs/EPSG/0/3395)
NAD27 (http://www.opengis.net/def/crs/EPSG/0/4267)
NAD27 (CRS27) (http://www.opengis.net/def/crs/OGC/1.3/CRS27)
NAD83 (http://www.opengis.net/def/crs/EPSG/0/4269)
NAD83 (CRS83) (http://www.opengis.net/def/crs/OGC/1.3/CRS83)
NAD83(CSRS) / New Brunswick Stereographic (http://www.opengis.net/def/crs/EPSG/0/2953)
NAD83(CSRS) / Prince Edward Isl. Stereographic (NAD83) (http://www.opengis.net/def/crs/EPSG/0/2954)
(expand to show entire list)

Links

The following resources are available for this collection:

- this collection as JSON
- this collection as XML
- the schema of feature instances of this collection
- the vector features of this collection (accepts query parameters for filtering, etc.)
- the CQL2 queryables of this collection
- perform complex feature queries within this collection
- the available styles for this collection, with links to style-specific map layers and tiles
- a map layer of this collection in the default style (accepts query parameters for subsetting, etc.)
- a legend graphic depicting the map layer of this collection in the default style
- map tiles of this collection in the default style

Copyright © 2023 MariaDB plc. Version 9.5.30.

Figure 18 — Accessing a layer through the Features API

The following code fragment shows the “Manhole” data being accessed from the MariaDB feature server as GeoJSON using the EPSG:27700 CRS:


```
{
  "type": "FeatureCollection",
  "timeStamp": "2023-11-07T14:44:44-05:00",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
    }
  },
  "links": [
    {
```

```
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=json&limit=10&offset=10",
      "rel": "next",
      "type": "application/geo+json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=json",
      "rel": "self",
      "type": "application/geo+json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/schemas/collection",
      "rel": "describedby",
      "type": "application/xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fgml%2Bxml",
      "rel": "alternate",
      "type": "application/gml+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fx-bxfs%2Bxml%3B%2
0version%3D1.0.0",
      "rel": "alternate",
      "type": "application/x-bxfs+xml; version=1.0.0"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fgml%2Bxml%3B%20ve
rsion%3D2.1",
      "rel": "alternate",
      "type": "application/gml+xml; version=2.1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fgml%2Bxml%3B%20ve
rsion%3D3.1",
      "rel": "alternate",
      "type": "application/gml+xml; version=3.1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=text%2Fxml%3B%20subType%3Dgml%2
F3.1.1%2Fprofiles%2Fgmlsf%2F1.0.0%2F0",
      "rel": "alternate",
      "type": "text/xml; subType=gml/3.1.1/profiles/gmlsf/1.0.0/0"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=text%2Fxml%3B%20subType%3Dgml%2
F3.1.1%2Fprofiles%2Fgmlsf%2F1.0.0%2F1",
      "rel": "alternate",
      "type": "text/xml; subType=gml/3.1.1/profiles/gmlsf/1.0.0/1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fgml%2Bxml%3B%20ve
rsion%3D3.2",
      "rel": "alternate",
```

```
      "type": "application/gml+xml; version=3.2"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fgml%2Bxml%3B%20ve
rsion%3D3.2%3B%20profile%3Dhttp%3A%2F%2Fwww.opengis.net%2Fdef%2Fprofile%2Fogc%2
F2.0%2Fgml-sf0",
      "rel": "alternate",
      "type": "application/gml+xml; version=3.2; profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf0"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fgml%2Bxml%3B%20ve
rsion%3D3.2%3B%20profile%3Dhttp%3A%2F%2Fwww.opengis.net%2Fdef%2Fprofile%2Fogc%2
F2.0%2Fgml-sf1",
      "rel": "alternate",
      "type": "application/gml+xml; version=3.2; profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Frss%2Bxml",
      "rel": "alternate",
      "type": "application/rss+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fatom%2Bxml",
      "rel": "alternate",
      "type": "application/atom+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fvnd.google-earth.
kml%2Bxml",
      "rel": "alternate",
      "type": "application/vnd.google-earth.kml+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fvnd.google-earth.
kmz",
      "rel": "alternate",
      "type": "application/vnd.google-earth.kmz"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=text%2Fhtml",
      "rel": "alternate",
      "type": "text/html"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fvnd.ogc.fg
%2Bjson",
      "rel": "alternate",
      "type": "application/vnd.ogc.fg+json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fvnd.ogc.fg%2Bjson
%3B%20conformance%3Dgeojson",
```

```
      "rel": "alternate",
      "type": "application/vnd.ogc.fg+json; conformance=geojson"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=application%2Fvnd.shp%2Boctet-
stream",
      "rel": "alternate",
      "type": "application/vnd.shp+octet-stream"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=applcation%2Fjson",
      "rel": "alternate",
      "type": "applcation/json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items?f=text%2Fxml",
      "rel": "alternate",
      "type": "text/xml"
    }
  ],
  "features": [
    {
      "type": "Feature",
      "id": "CWFID.Manhole.0.1",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -0.1048996567,
          51.51406059
        ]
      },
      "properties": {
        "ANCILLARYR": 0,
        "ENABLED": 1,
        "DATECREATE": "2017-03-16T00:00:00",
        "DATEMODIFI": "2017-03-16T00:00:00",
        "GISID": 9580013,
        "WATERTRACE": 0,
        "MAINTAINER": "Sample Sewer Network",
        "YEARBUILTC": "Z",
        "SYMBOLROTA": 0,
        "TRUNK": "N",
        "LOCATIONCH": "EI",
        "SUBTYPECD": 4,
        "PURPOSE": "F",
        "LIFECYCLES": "L",
        "COVERLEVEL": 0,
        "INVERTLEVE": 0,
        "DEPTHINM": 0,
        "METHODOFAB": "N",
        "SIDEENTRY": "N",
        "COVERVENTE": "N",
        "SUMMIT": "N",
        "HSCLASSIFI": "N",
        "CONSTRUCTI": "Z",
        "STEPSORLAD": "Z",
        "COVERSHAPE": "Z",
        "COVERDUTY": "Z",
        "OPENINGDIA": 0,
        "OPENINGWID": 0,
```

```
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 15,
      "FLUSHINGPO": "N",
      "SHAPEX": 531596.966932,
      "SHAPEY": 181161.447554,
      "LEGISLATIO": "N",
      "DATEPOSTED": "2017-03-17T00:00:00",
      "TWTRACEWEI": 14803,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.2",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1049322282,
        51.51408668
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2017-03-16T00:00:00",
      "DATEMODIFI": "2017-03-16T00:00:00",
      "GISID": 9580023,
      "WATERTRACE": 0,
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 0,
      "TRUNK": "N",
      "LOCATIONCH": "EI",
      "SUBTYPECD": 4,
      "PURPOSE": "F",
      "LIFECYCLES": "L",
      "COVERLEVEL": 0,
      "INVERTLEVE": 0,
      "DEPTHINM": 0,
      "METHODOFAB": "N",
      "SIDEENTRY": "N",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "Z",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
```

```
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 15,
      "FLUSHINGPO": "N",
      "SHAPEX": 531594.631761,
      "SHAPEY": 181164.29037,
      "LEGISLATIO": "N",
      "DATEPOSTED": "2017-03-17T00:00:00",
      "TWTRACEWEI": 14803,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.3",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1041095606,
        51.51318858
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2008-08-23T00:00:00",
      "GISID": 6236612,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31816007",
      "ALTERNATER": 3181,
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 195,
      "TRUNK": "Y",
      "LOCATIONCH": "UC",
      "SUBTYPECD": 1,
      "PURPOSE": "SO",
      "LIFECYCLES": "L",
      "COVERLEVEL": 0,
      "INVERTLEVE": 0,
      "DEPTHINM": 0,
      "METHODOFAB": "N",
      "SIDEENTRY": "Y",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "N",
```

```
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 8,
      "FLUSHINGPO": "N",
      "SHAPEX": 531654.3,
      "SHAPEY": 181065.9,
      "LEGISLATIO": "Z",
      "DATEPOSTED": "2016-08-12T00:00:00",
      "TWTRACEWEI": 13493,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.4",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1040584795,
        51.51320394
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2008-08-23T00:00:00",
      "GISID": 6236623,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31816009",
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 195,
      "TRUNK": "Y",
      "LOCATIONCH": "UC",
      "SUBTYPECD": 1,
      "PURPOSE": "SO",
      "LIFECYCLES": "L",
      "COVERLEVEL": 0,
      "INVERTLEVE": 0,
      "DEPTHINM": 0,
      "METHODOFAB": "N",
      "SIDEENTRY": "Y",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
```

```
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "N",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 8,
      "FLUSHINGPO": "N",
      "SHAPEX": 531657.8,
      "SHAPEY": 181067.7,
      "LEGISLATIO": "Z",
      "DATEPOSTED": "2016-08-12T00:00:00",
      "TWTRACEWEI": 13493,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.5",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1044895631,
        51.51601126
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2017-08-31T00:00:00",
      "GISID": 6237017,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31816303",
      "ALTERNATER": 6303,
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 192,
      "TRUNK": "N",
      "LOCATIONCH": "EI",
      "SUBTYPECD": 1,
      "PURPOSE": "C",
      "LIFECYCLES": "L",
      "COVERLEVEL": 6.26,
      "INVERTLEVE": 2.5,
      "DEPTHINM": 3.76,
      "METHODOFAB": "N",
      "SIDEENTRY": "N",
```

```
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "N",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 1,
      "FLUSHINGPO": "N",
      "SHAPEX": 531619.8,
      "SHAPEY": 181379.1,
      "LEGISLATIO": "T105A",
      "DATEPOSTED": "2017-09-05T00:00:00",
      "TWTRACEWEI": 13493,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.6",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1041303778,
        51.51605312
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2017-08-31T00:00:00",
      "GISID": 6240284,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31816302",
      "ALTERNATER": 6302,
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 191,
      "TRUNK": "N",
      "LOCATIONCH": "UC",
      "SUBTYPECD": 1,
      "PURPOSE": "C",
      "LIFECYCLES": "L",
      "COVERLEVEL": 7.24,
      "INVERTLEVE": 3.43,
```

```
      "DEPTHINM": 3.81,
      "METHODOFAB": "N",
      "SIDEENTRY": "N",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "N",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 1,
      "FLUSHINGPO": "N",
      "SHAPEX": 531644.6,
      "SHAPEY": 181384.4,
      "LEGISLATIO": "T105A",
      "DATEPOSTED": "2017-09-05T00:00:00",
      "TWTRACEWEI": 13493,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.7",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1001826388,
        51.51344357
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2015-01-22T00:00:00",
      "DATEMODIFI": "2015-01-22T00:00:00",
      "GISID": 8796230,
      "WATERTRACE": 0,
      "REFERENCE": "TQ3181911A",
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 0,
      "TRUNK": "N",
      "LOCATIONCH": "EI",
      "SUBTYPECD": 4,
      "PURPOSE": "F",
      "LIFECYCLES": "L",
```

```
      "COVERLEVEL": 0,
      "INVERTLEVE": 0,
      "DEPTHINM": 0,
      "METHODOFAB": "N",
      "SIDEENTRY": "N",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "Z",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 15,
      "FLUSHINGPO": "N",
      "SHAPEX": 531926.033793,
      "SHAPEY": 181101.320665,
      "LEGISLATIO": "N",
      "DATEPOSTED": "2015-01-27T00:00:00",
      "TWTRACEWEI": 14803,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.8",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.105375955,
        51.51330347
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2008-08-23T00:00:00",
      "GISID": 6236188,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31815006",
      "ALTERNATER": 5006,
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 189,
      "TRUNK": "N",
      "LOCATIONCH": "UC",
```

```
      "SUBTYPECD": 1,
      "PURPOSE": "C",
      "LIFECYCLES": "L",
      "COVERLEVEL": 10.13,
      "INVERTLEVE": 6.6,
      "DEPTHINM": 3.53,
      "METHODOFAB": "N",
      "SIDEENTRY": "N",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "N",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 1,
      "FLUSHINGPO": "N",
      "SHAPEX": 531566.1,
      "SHAPEY": 181076.4,
      "LEGISLATIO": "Z",
      "DATEPOSTED": "2013-09-25T00:00:00",
      "TWTRACEWEI": 13493,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.9",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1073535056,
        51.51395949
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2007-04-12T00:00:00",
      "GISID": 6242448,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31814102",
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 191,
```

```
      "TRUNK": "N",
      "LOCATIONCH": "UC",
      "SUBTYPECD": 1,
      "PURPOSE": "A",
      "LIFECYCLES": "L",
      "COVERLEVEL": 0,
      "INVERTLEVE": 0,
      "DEPTHINM": 0,
      "METHODOFAB": "N",
      "SIDEENTRY": "N",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "N",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 10,
      "FLUSHINGPO": "N",
      "SHAPEX": 531427,
      "SHAPEY": 181145.8,
      "LEGISLATIO": "Z",
      "TWTRACEWEI": 13493,
      "MANHOLECOV": "N"
    }
  },
  {
    "type": "Feature",
    "id": "CWFID.Manhole.0.10",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.1110143471,
        51.51099782
      ]
    },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2008-08-23T00:00:00",
      "GISID": 6243485,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31801802",
      "ALTERNATER": 1802,
      "MAINTAINER": "Sample Sewer Network",
```

```
        "YEARBUILTC": "Z",
        "SYMBOLROTA": 272,
        "TRUNK": "N",
        "LOCATIONCH": "UC",
        "SUBTYPECD": 1,
        "PURPOSE": "C",
        "LIFECYCLES": "L",
        "COVERLEVEL": 4.49,
        "INVERTLEVE": 0,
        "DEPTHINM": 0,
        "METHODOFAB": "N",
        "SIDEENTRY": "N",
        "COVERVENTE": "N",
        "SUMMIT": "N",
        "HSCLASSIFI": "N",
        "CONSTRUCTI": "Z",
        "STEPSORLAD": "N",
        "COVERSHAPE": "Z",
        "COVERDUTY": "Z",
        "OPENINGDIA": 0,
        "OPENINGWID": 0,
        "SHAFTDIAME": 0,
        "SHAFTWIDTH": 0,
        "SHAFTDEPTH": 0,
        "CHAMBERDIA": 0,
        "CHAMBERWID": 0,
        "GALLERYDIA": 0,
        "GALLERYWID": 0,
        "BIFURCATIO": "N",
        "DEMARCATIO": "N",
        "CATCHPIT": "N",
        "CATCHPITDE": 0,
        "GPSX": 0,
        "GPSY": 0,
        "GPSZ": 0,
        "DEEPBORESO": "N",
        "SYMBOLCODE": 1,
        "FLUSHINGPO": "N",
        "SHAPEX": 531181.5,
        "SHAPEY": 180809.9,
        "LEGISLATIO": "Z",
        "TWTRACEWEI": 13493,
        "MANHOLECOV": "N"
      }
    }
  ]
}
```


The following fragment shows one instance of a “Manhole” feature being presented as JSON-FG using CRS EPSG-27700 from the MariaDB feature server:


```
{
  "type": "Feature",
  "conformsTo" : [ "[ogc-json-fg-1-0.2:core]",
                   "[ogc-json-fg-1-0.2:types-schemas]" ],
  "featureType": "Manhole",
  "featureSchema": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/
London_2023/ogcapi/Store_4/collections/Manhole/schema",
  "id": "CWFID.Manhole.0.10",
  "coordRefSys": "http://www.opengis.net/def/crs/EPSG/0/27700",
  "geometry": null,
  "place": {
    "type": "Point",
```

```
      "coordinates": [
        531181.5,
        180809.9
      ]
    },
    "time": { "timestamp": "2008-08-23T00:00:00Z" },
    "properties": {
      "ANCILLARYR": 0,
      "ENABLED": 1,
      "DATECREATE": "2008-08-23T00:00:00",
      "DATEMODIFI": "2008-08-23T00:00:00",
      "GISID": 6243485,
      "WATERTRACE": 0,
      "REFERENCE": "TQ31801802",
      "ALTERNATER": 1802,
      "MAINTAINER": "Sample Sewer Network",
      "YEARBUILTC": "Z",
      "SYMBOLROTA": 272,
      "TRUNK": "N",
      "LOCATIONCH": "UC",
      "SUBTYPECD": 1,
      "PURPOSE": "C",
      "LIFECYCLES": "L",
      "COVERLEVEL": 4.49,
      "INVERTLEVE": 0,
      "DEPTHINM": 0,
      "METHODOFAB": "N",
      "SIDEENTRY": "N",
      "COVERVENTE": "N",
      "SUMMIT": "N",
      "HSCLASSIFI": "N",
      "CONSTRUCTI": "Z",
      "STEPSORLAD": "N",
      "COVERSHAPE": "Z",
      "COVERDUTY": "Z",
      "OPENINGDIA": 0,
      "OPENINGWID": 0,
      "SHAFTDIAME": 0,
      "SHAFTWIDTH": 0,
      "SHAFTDEPTH": 0,
      "CHAMBERDIA": 0,
      "CHAMBERWID": 0,
      "GALLERYDIA": 0,
      "GALLERYWID": 0,
      "BIFURCATIO": "N",
      "DEMARCATIO": "N",
      "CATCHPIT": "N",
      "CATCHPITDE": 0,
      "GPSX": 0,
      "GPSY": 0,
      "GPSZ": 0,
      "DEEPBORESO": "N",
      "SYMBOLCODE": 1,
      "FLUSHINGPO": "N",
      "SHAPEX": 531181.5,
      "SHAPEY": 180809.9,
      "LEGISLATIO": "Z",
      "TWTRACEWEI": 13493,
      "MANHOLECOV": "N"
    },
    "links": [
      {
```

```
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4",
      "rel": "service,"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole?f=application%2Fjson",
      "rel": "collection",
      "type": "application/json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole?f=application%2Fxml",
      "rel": "collection",
      "type": "application/xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/schemas/collection",
      "rel": "describedby",
      "type": "application/xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fvnd.ogc.fg%2Bjson",
      "rel": "self",
      "type": "application/vnd.ogc.fg+json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fgml%2Bxml",
      "rel": "alternate",
      "type": "application/gml+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fx-bxfs%2Bxml%3B%20version%3D1.0.0",
      "rel": "alternate",
      "type": "application/x-bxfs+xml; version=1.0.0"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fgml%2Bxml%3B%20version%3D2.1",
      "rel": "alternate",
      "type": "application/gml+xml; version=2.1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fgml%2Bxml%3B%20version%3D3.1",
      "rel": "alternate",
      "type": "application/gml+xml; version=3.1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=text%2Fxml%3
B%20subType%3Dgml%2F3.1.1%2Fprofiles%2Fgmlsf%2F1.0.0%2F0",
      "rel": "alternate",
      "type": "text/xml; subType=gml/3.1.1/profiles/gmlsf/1.0.0/0"
```

```
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=text%2Fxml%3
B%20subType%3Dgml%2F3.1.1%2Fprofiles%2Fgmlsf%2F1.0.0%2F1",
      "rel": "alternate",
      "type": "text/xml; subType=gml/3.1.1/profiles/gmlsf/1.0.0/1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fgml%2Bxml%3B%20version%3D3.2",
      "rel": "alternate",
      "type": "application/gml+xml; version=3.2"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fgml%2Bxml%3B%20version%3D3.2%3B%20profile%3Dhttp%3A%2F%2Fwww.opengis.net%2Fd
ef%2Fprofile%2Fogc%2F2.0%2Fgml-sf0",
      "rel": "alternate",
      "type": "application/gml+xml; version=3.2; profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf0"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fgml%2Bxml%3B%20version%3D3.2%3B%20profile%3Dhttp%3A%2F%2Fwww.opengis.net%2Fd
ef%2Fprofile%2Fogc%2F2.0%2Fgml-sf1",
      "rel": "alternate",
      "type": "application/gml+xml; version=3.2; profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf1"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Frss%2Bxml",
      "rel": "alternate",
      "type": "application/rss+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fatom%2Bxml",
      "rel": "alternate",
      "type": "application/atom+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fvnd.google-earth.kml%2Bxml",
      "rel": "alternate",
      "type": "application/vnd.google-earth.kml+xml"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fvnd.google-earth.kmz",
      "rel": "alternate",
      "type": "application/vnd.google-earth.kmz"
    },
    {
```

```
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=text
%2Fhtml",
      "rel": "alternate",
      "type": "text/html"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fgeo%2Bjson",
      "rel": "alternate",
      "type": "application/geo+json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=application
%2Fvnd.shp%2Boctet-stream",
      "rel": "alternate",
      "type": "application/vnd.shp+octet-stream"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=applcation
%2Fjson",
      "rel": "alternate",
      "type": "applcation/json"
    },
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023/ogcapi/Store_4/collections/Manhole/items/CWFID.Manhole.0.10?f=text%2Fxml",
      "rel": "alternate",
      "type": "text/xml"
    }
  ]
}
```


The MUDDI data was loaded from SHAPE files and as such does not conform to the MUDDI schema. The MariaDB feature server has the ability to transform the internal schema to an external schema on-the-fly and work was begun at the code sprint to create the mapping files to transform the stored collections into conformant MUDDI GML and GeoJSON responses. However, time constraints prevented the work from being completed during the code sprint.

4.3.1.2.2. JSON-FG

There were several aspects of MariaDB's Feature server JSON-FG output that did not conform to the latest editor's draft of the JSON-FG specification and these issues were addressed during the code sprint. Specifically, the MariaDB server did not previously generate the "conformsTo," "featureType," and "featureSchema" members. These members can now be seen in the JSON-FG example above.

4.3.1.2.3. OGC API — Records

Discussion topics

Two main topics were discussed in the OGC API — Records breakout session.

- The first concerned “faceted” searching (aka Aggregations) which resulted in a pull request being created for the proposal. The decision was made that this capability would not be part of Part 1 — Core but would be in a stand-alone part.
- The second topic of discussion was federated searching. The conclusion of the discussion was that no special changes are required to Part 1 — Core to support federated searching. A federated search capability can be easily implemented on either the client side or the server side by simply querying N catalogues (perhaps simultaneously) and then aggregating the results.

Implementation

Issue 275 from the OGC API — Records SWG discusses templated links and how unaware clients, trying to resolve a templated link, would encounter errors because the unaware client would not know to replace the substitution variables in the template. This issue presented several approaches to resolving the problem which were discussed in a subsequent OGC Architecture DWG meeting convened to agree on an OGC-wide approach for handling templated links. At that meeting it was agreed that the approach described in Pull request 290 was most consistent with current practices on the Web.

During the code sprint, the MariaDB OGC API — Records implementation was updated to handle templated links as described in Pull request 290. Below is a sample record containing a templated links section:


```
{
  "id": "urn:uuid:9c953c1e-7dd8-11ee-907c-6b42beb7f5d4",
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [ [ -0.1154702047, 51.50711155 ],
        [ -0.09681305432, 51.50711155 ],
        [ -0.09681305432, 51.51830892 ],
        [ -0.1154702047, 51.51830892 ],
        [ -0.1154702047, 51.50711155 ] ] ]
  },
  "properties": {
    "type": "urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:
Service",
    "title": "London 2023 MariaDB Geospatial WMS",
    "description": "OGC-compliant cascading web map server (WMS) by MariaDB.",
    "baseUrl": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023",
    "urn:cw:def:ebRIM-SlotName:cubeVersion": "9.5.30",
    "layerLimit": "-1",
    "maxHeight": "3000",
    "maxWidth": "4000",
    "onlineResource": "https://mariadb.com/products/skysql/geospatial/",
    "resourceType": "http://www.opengis.net/wms"
  },
  "linkTemplates": [
    {
      "href": "https://dev.cubewerx.com/~mgalluch/mariadbgeo/cubeserv/London_
2023?SERVICE=WMS&REQUEST=GetMap&layer={layerName}&bbox={minX},{minY},{maxX},
{maxY}&crs={crs}&width={width}&height={height}",
      "rel": "describes",
      "title": "London 2023 MariaDB Geospatial WMS"
```

```
    }
  ],
  "links": [
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues",
      "rel": "service",
      "title": "service offering this record"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest",
      "rel": "collection",
      "title": "collection containing this record"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=json&s=ogcapirecord",
      "rel": "self",
      "type": "application/geo+json",
      "title": "this record"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/ogcApiForRecord/urn%3Auuid%3A9c953c1e-7dd8-11ee-
907c-6b42beb7f5d4",
      "rel": "ogcApiForRecord",
      "title": "dynamic OGC API langing page"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=xml&s=ogcapirecord",
      "rel": "alternate",
      "type": "application/xml",
      "title": "this record using the \"OGC API Record\" schema encoded as xml"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=html&s=ogcapirecord",
      "rel": "alternate",
      "type": "text/html",
      "title": "this record using the \"OGC API Record\" schema encoded as
html"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=xml&s=cswrecord30",
      "rel": "alternate",
      "type": "application/csw30+xml",
      "title": "this record using the \"CSW Record v3.0\" schema encoded as
xml"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=xml&s=cswrecord20",
      "rel": "alternate",
      "type": "application/csw20+xml",
```

```
      "title": "this record using the \"CSW Record v2.X\" schema encoded as
xml"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=xml&s=ebrim",
      "rel": "alternate",
      "type": "application/ebrim+xml",
      "title": "this record using the \"ebRIM\" schema encoded as xml"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=xml&s=atom",
      "rel": "alternate",
      "type": "application/atom+xml",
      "title": "this record using the \"ATOM\" schema encoded as xml"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9c953c1e-7dd8-11ee-907c-
6b42beb7f5d4?f=xml&s=rss",
      "rel": "alternate",
      "type": "application/rss+xml",
      "title": "this record using the \"RSS\" schema encoded as xml"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da91a9e-7dd8-11ee-8622-
4fd1cb3998dc?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "End Item"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9daa2b96-7dd8-11ee-9739-
2becf9c93847?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Fitting"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9dab2dca-7dd8-11ee-89ec-
53b042fc578f?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Hydrant"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9dac8c88-7dd8-11ee-8070-
a37d1bf0c23b?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Network Meter"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9dad9ad8-7dd8-11ee-8012-
2fbe67e88d7f?f=json",
```

```
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Network Opt Valve"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9daeadf6-7dd8-11ee-8066-
1ffca5339551?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Pressure Cont Valve"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9dafb052-7dd8-11ee-960f-
57d7e523dc01?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Trunk Main"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d7e40a8-7dd8-11ee-acbf-
6f5fdd053539?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Non Flow Equipment"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d7f4e26-7dd8-11ee-a50e-
234b32530474?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Pipe"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d809678-7dd8-11ee-bd80-
071461ba6f4a?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "IDNO"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d813ec0-7dd8-11ee-be07-
3b6b398204e4?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Electric Cable Line"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d81b8c8-7dd8-11ee-a6fe-
f322135c7733?f=json",
      "type": "application/geo+json",
      "title": "IDNO"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d827d44-7dd8-11ee-881c-
9737ccd6b355?f=json",
```

```
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Electric Demand Point"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d83cab4-7dd8-11ee-84e9-
6ff4debe02df?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Site Polygon"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8517fc-7dd8-11ee-872a-
0733383085f9?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Substation Point"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d866652-7dd8-11ee-b77d-
a3196972ffcc?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Local Authority"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d870e2c-7dd8-11ee-81d4-
af98f059b7b2?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Bollards"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8798b0-7dd8-11ee-b1a7-
93a783613103?f=json",
      "type": "application/geo+json",
      "title": "LocalAuthority"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8873e8-7dd8-11ee-9f81-
736ce96b9dc0?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Drains"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d89c27a-7dd8-11ee-8507-
534bf09a4a39?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Street Lights"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8b179c-7dd8-11ee-b6e6-
577d4ef8dfca?f=json",
```

```
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Subway Lighting"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8c6534-7dd8-11ee-b6cb-
2fe8446cd655?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Sewer"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8d0818-7dd8-11ee-8388-
237dac12edb7?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Abandoned Point"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8d828e-7dd8-11ee-87a5-
97dde6796383?f=json",
      "type": "application/geo+json",
      "title": "Sewer"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8e457a-7dd8-11ee-a7cf-
1b9af63148f3?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Abandoned Sewer"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d8fac4e-7dd8-11ee-b5bc-
cba7512a8434?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Ancillary"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d9107e2-7dd8-11ee-8815-
f3e020018ef9?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Chamber"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d9252a0-7dd8-11ee-8d43-
334e768efe4f?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Control Valve"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d935c7c-7dd8-11ee-ae63-
5fb06c3abc1e?f=json",
```

```
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Drain"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d94aece-7dd8-11ee-8dab-
7fe7fe7c3cac?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Duct or Crossing"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d95fffe-7dd8-11ee-be43-
9b3fd957c948?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "End Sewer Item"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d97615a-7dd8-11ee-8558-
1329ebca4654?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Fitting"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d98aae2-7dd8-11ee-8374-
afa41f426eae?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Gravity Sewer"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d99f1ea-7dd8-11ee-8b32-
97c352f2e781?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Invert Level Point"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d9b50da-7dd8-11ee-a9b1-
df9ffd263109?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Lateral"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d9c4dbe-7dd8-11ee-ac47-
df2fbfafa62e?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Manhole"
    },
    {
```

```
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d9d99a8-7dd8-11ee-8650-
330f1e821069?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Meter"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d9ee970-7dd8-11ee-a706-
e31bbb18d5f0?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Operational Site"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da04770-7dd8-11ee-a36d-
ebe680b57c48?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Water"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da0f440-7dd8-11ee-bb4c-
131fba7f39c3?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Abandoned Water Device"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da20416-7dd8-11ee-908f-
f3228e168f63?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Abandoned Water Main"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da27ebe-7dd8-11ee-b8a8-
9b7786b75552?f=json",
      "type": "application/geo+json",
      "title": "Water"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da346fa-7dd8-11ee-af0d-
6bd5b47cd648?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Chamber"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da448ac-7dd8-11ee-91d5-
4386a1d007cd?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Connection Main"
    },
    {
```

```
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da59d38-7dd8-11ee-ade9-
672cc04f5409?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Control Pillar"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da6a156-7dd8-11ee-90de-
07054a3b15c0?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Distribution Main"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9da7f808-7dd8-11ee-a609-
c788d0aee706?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Duct"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9ccb2860-7dd8-11ee-9e88-
d7a3e25b7bd0?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "CubeSERV WMS layers"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d6166f4-7dd8-11ee-aade-
b794a356708b?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Electricity"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d63363c-7dd8-11ee-89e7-
af9e397b43a0?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Cross Section"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d63fe78-7dd8-11ee-a7b4-
f30a06ab116e?f=json",
      "type": "application/geo+json",
      "title": "Basic style"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d65882e-7dd8-11ee-b0dd-
e3dbf2e31bc7?f=json",
      "type": "application/geo+json",
      "title": "Electricity"
    },
    {
```

```
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d66e7b4-7dd8-11ee-aad7-
db1fb54aba79?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Cross Section Arrow"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d706d52-7dd8-11ee-bbd0-
7ffa6ee86150?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "LV Link Box"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d71772e-7dd8-11ee-9523-
3389c1e88f82?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Pit"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d72e7ee-7dd8-11ee-9d0c-
775756b48335?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Primary Substation Building D"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d745926-7dd8-11ee-a9ed-
dfb31b2b4b58?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Service Cable"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d75a47a-7dd8-11ee-bbd9-
772c85f6d250?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Substation"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d77004a-7dd8-11ee-bb45-
b7db6c416fbd?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Supply Point"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d78738a-7dd8-11ee-b50e-
db6a7bc93333?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Gas"
    },
```

```
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d791d1c-7dd8-11ee-ad2e-
d34e158bb8cc?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Abandoned Pipe"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d79985a-7dd8-11ee-87b8-
37e490c862c6?f=json",
      "type": "application/geo+json",
      "title": "Gas"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d7a6262-7dd8-11ee-ab82-
b3272dabe4dd?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Depth of Cover"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d7babd6-7dd8-11ee-a184-
9b6abd3f47b3?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "End Closure"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d7cf284-7dd8-11ee-8301-
9bbcbe2ce4c9?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Flow Equipment"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d68d894-7dd8-11ee-bf49-
bbe7acc7f0a0?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "Duct"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d69d596-7dd8-11ee-b19b-
23881c04eb74?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "EHV Cable"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d6b29b4-7dd8-11ee-b533-
d7d9e4f2c138?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "HV Cable"
    },
```

```
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d6c7396-7dd8-11ee-acc6-
0b71b22e270a?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "HV Joint"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d6dc002-7dd8-11ee-8007-
6fbba4a4737a?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "LV Cable"
    },
    {
      "href": "https://www.pvretano.com/cubewerx/cubeserv/default/ogcapi/
catalogues/collections/wrstest/items/urn%3Auuid%3A9d6f1420-7dd8-11ee-ac16-
d7aabcec602c?f=json",
      "rel": "operatesOn",
      "type": "application/geo+json",
      "title": "LV Joint"
    }
  ]
}
```


4.3.1.2.4. OGC API — Processes

Discussion topics

The OGC API — Processes — Part 1: Core Standard defines two ways that a server can use to notify a client that an asynchronously executed process has completed execution. The first method involves polling using the GetStatus operation. The second method involves OpenAPI callbacks. This topic was discussed during a breakout session at the code sprint. The approach involves applying the Publish-Subscribe workflow used by OGC API — EDR. It was agreed that, subject to SWG approval, AsyncAPI and the PubSub workflow should be offered as a notification option in addition the current polling and callback mechanisms (which should be preserved for backward compatability).

Implementation

During the code sprint, changes were made to the MariaDB OGC API — Processes server to align with editor's drafts of Part 1 and Part 2. The following images illustrate the MariaDB thematic exploitation test harness using the MariaDB OGC API — Records and OGC API — Processes servers updated during the code sprint.

A series of screenshots illustrating interaction with the system are shown in the following figures. Figure 19 shows the TEP accessing a catalog to locate images of interest.

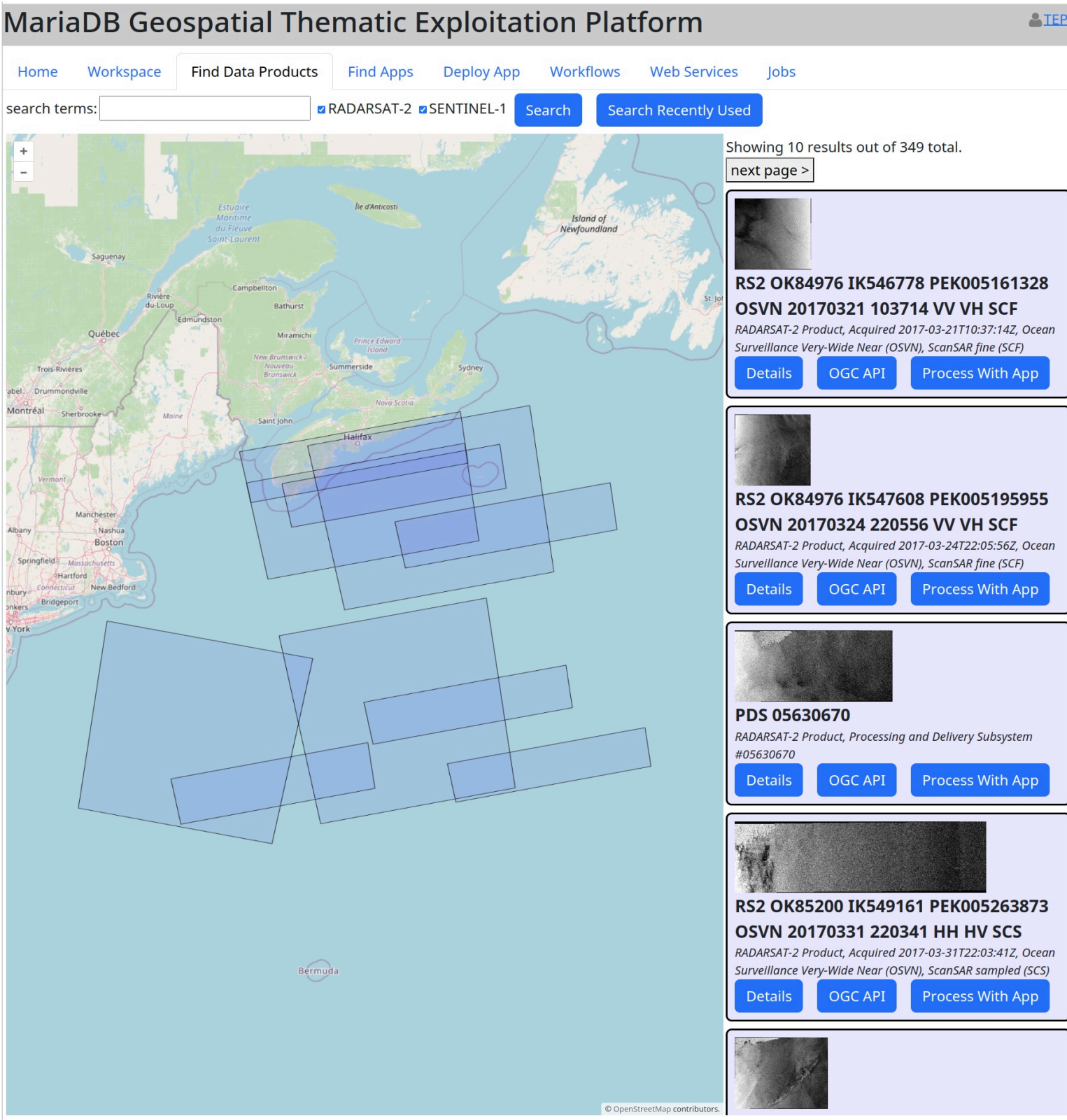


Figure 19 — Screenshots of the TEP accessing the catalog to locate images of interest

Figure 20 illustrates how the user selects one of the images and how the catalog is accessed to find a compatible process (the Ship Detection Process in this case).

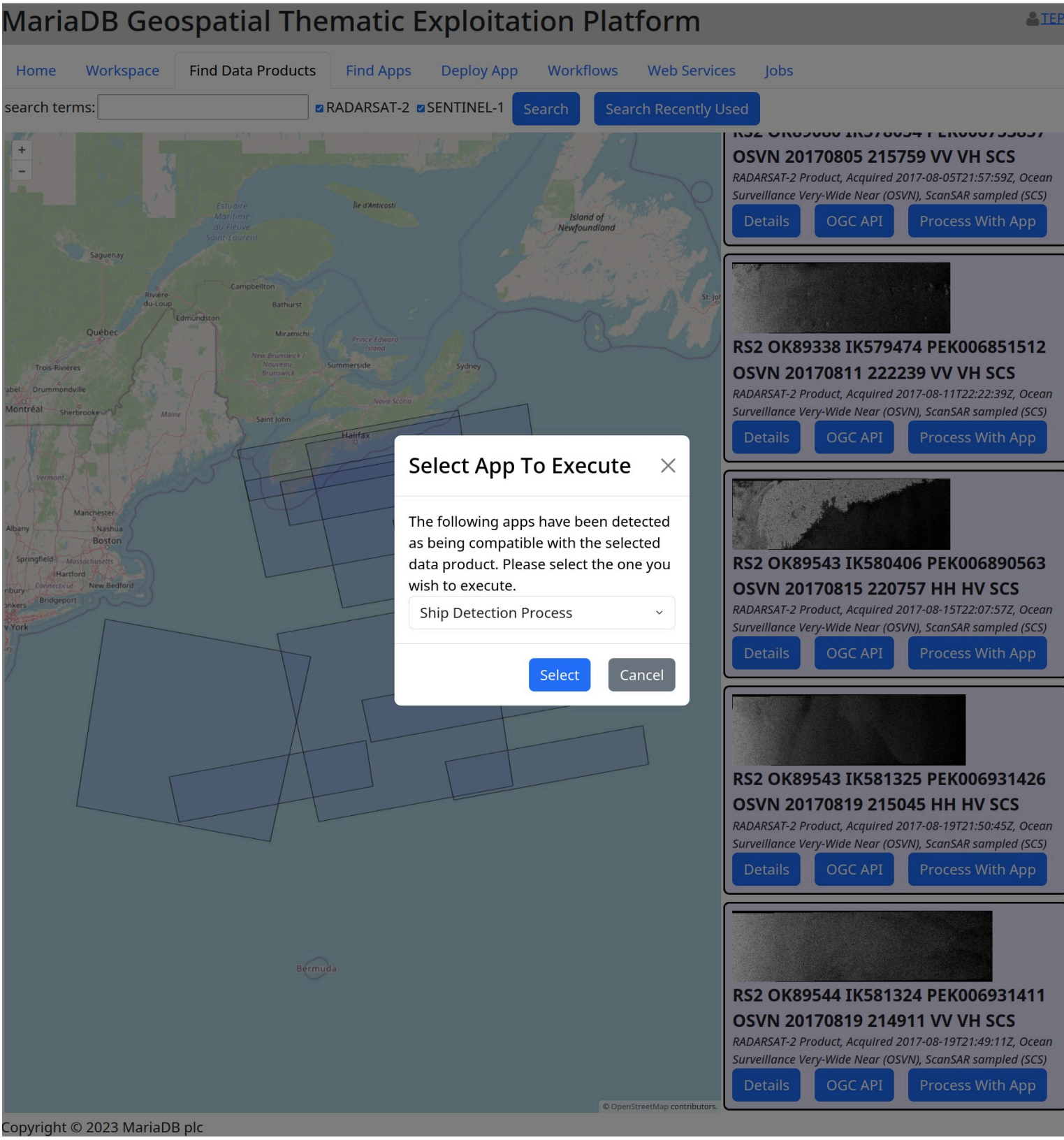


Figure 20 — Selection of an image and compatible process

Figure 21 illustrates process execution (top-left), job acceptance (top-right), job status with an executed process completed successfully (lower-left), and display of result (lower-right).

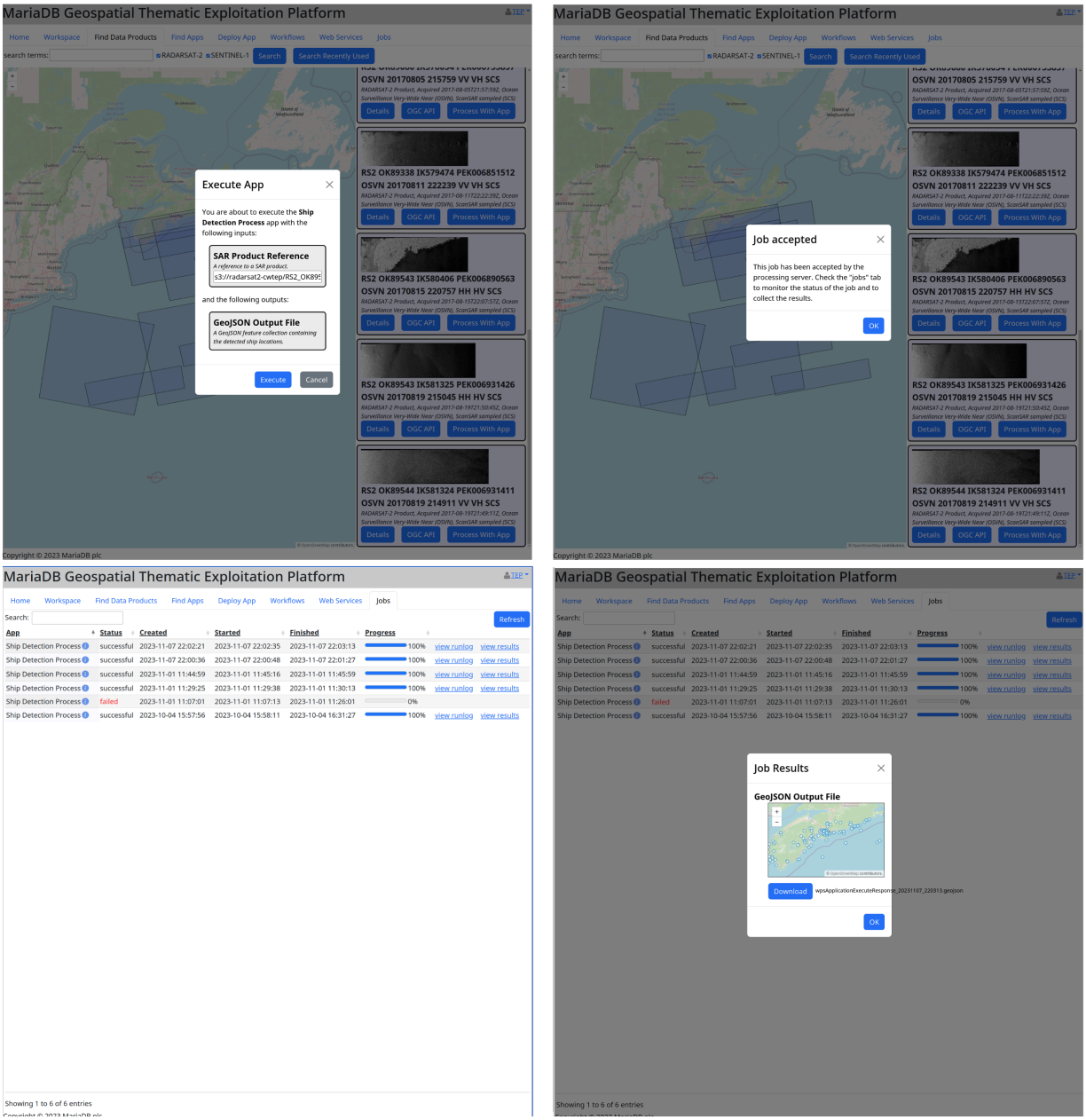


Figure 21 — Screenshots of process execution (top-left), job acceptance (top-right), job status (lower-left), and display of result (lower-right)

4.3.2. ldproxy

Sprint participants from interactive instruments worked on upgrades of several ldproxy modules during the sprint. The modules included support for JSON-FG, Feature Schemas, and PubSub. The upgrades are documented in the next sub-sections.

The upgraded version of ldproxy was used in the following two deployments.

1. The ldproxy demonstration deployment at demo.ldproxy.net. The deployment and the APIs in the deployment are documented in the ldproxy/demo GitHub repository. All APIs supporting OGC API — Features used the upgraded modules for JSON-FG and Feature Schemas.

2. A new deployment for the sprint at cs2310.ldproxy.net. This deployment provides access to the Daraa dataset that is also available in the ldproxy demonstration deployment. The data is derived from OpenStreetMap in the region of Daraa, Syria, converted by NGA to NGA's Topographic Data Store schema. This deployment also adds support for OGC API — Features — Part 4: Create, Replace, Update, and Delete, as well as authentication using OpenID Connect and role-based access control through OAuth2 scopes. The deployment also supports PubSub capabilities. The deployment is described below to illustrate the additional capabilities.

4.3.2.1. Support for JSON-FG 0.2.1

During the sprint, multiple pull requests were created to upgrade ldproxy to the latest JSON-FG draft version 0.2.1. The changes in Pull Request 1094 are:

- add "conformsTo";
- add "featureSchema";
- add new option "useCuries" to write Safe CURIEs instead of URIs; and
- deprecate "describedby."

The pull request was merged in the days after the sprint once the internal review process was completed.

4.3.2.2. Support for OGC API — Features — Part 5: Schemas

The changes in Pull Request 1093, which was created during the code sprint, implemented the edits to the specification discussed in the OGC API — Features breakout session. The changes include:

- add support for "x-ogc-unit" (rename from "unit", which previously was an ldproxy extension);
- restrict to JSON Schema 2020-12; and
- fix an error in the link relation type ("schema" instead of "schema-item").

The pull request was merged in the days after the code sprint once the internal review process was completed.

4.3.2.3. Support for PubSub

As part of OGC Testbed 19, interactive instruments is adding a new ldproxy module for the planned OGC API PubSub standard with a focus on subscriptions on new, updated, or deleted features.

During the PubSub implementation in the testbed, two issues were identified and discussed during an OGC API PubSub breakout during the sprint: channel identifiers and links are not useful / usable.

A new branch pubsub3 was created for the changes during the sprint.

The cs2310.ldproxy.net deployment was configured to support notifications for changes to features in the Daraa dataset.

The PubSub capabilities are described in an AsyncAPI definition document. Links to both the OpenAPI and AsyncAPI definition documents are clearly visible on the landing page, as shown in the following figure.

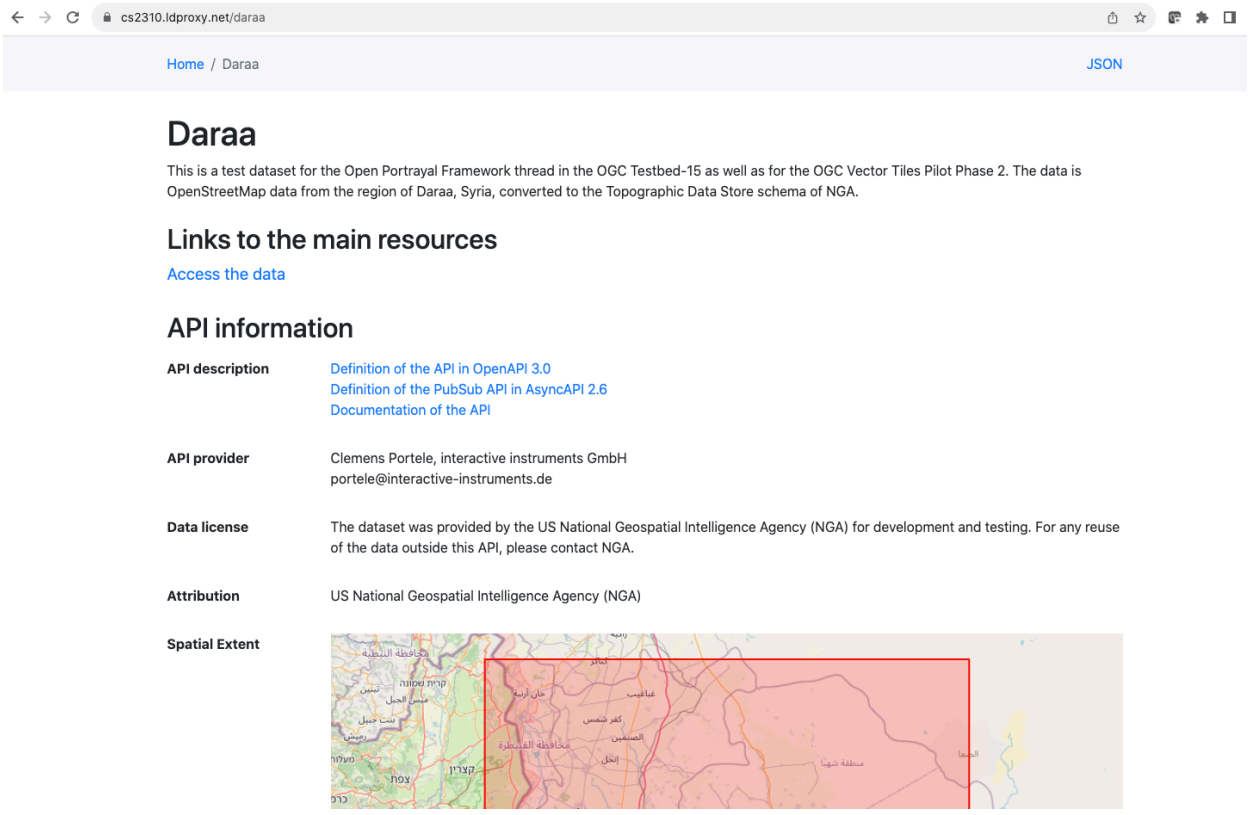


Figure 22 — ldproxy landing page

Below is a fragment of the AsyncAPI definition document created to describe the supported PubSub capability. The fragment shows two of the thirty channels created for monitoring changes on feature collections provided by the API.

```
{
```

```
  "info" : {
    "title" : "Daraa",
    "description" : "This is a test dataset for the Open Portrayal Framework
thread in the OGC Testbed-15 as well as for the OGC Vector Tiles Pilot Phase
2. The data is OpenStreetMap data from the region of Daraa, Syria, converted
to the Topographic Data Store schema of NGA.",
    "version" : "1.0.0",
    "contact" : {
      "name" : "Clemens Portele, interactive instruments GmbH",
      "email" : "portele@interactive-instruments.de"
    },
    "license" : {
      "name" : "The dataset was provided by the US National Geospatial
Intelligence Agency (NGA) for development and testing. For any reuse of the
data outside this API, please contact NGA."
    }
  },
  "servers" : {
    "t19" : {
      "protocol" : "secure-mqtt",
      "protocolVersion" : "3.1.1",
      "url" : "t19.ldproxy.net:8883",
      "bindings" : {
        "clientId" : "cs2310.ldproxy.net",
        "cleanSession" : true,
        "bindingVersion" : "0.1.0"
      }
    }
  },
  "channels" : {
    "ogcapi/cs2310.ldproxy.net/daraa/collections/AeronauticCrv/items" : {
      "subscribe" : {
        "operationId" : "featureChange_AeronauticCrv_items",
        "bindings" : {
          "qos" : 0,
          "retain" : false,
          "bindingVersion" : "0.1.0"
        },
        "message" : {
          "$ref" : "#/components/messages/featureChange_AeronauticCrv"
        }
      },
      "servers" : [ "t19" ]
    },
    "ogcapi/cs2310.ldproxy.net/daraa/collections/AeronauticSrf/items" : {
      "subscribe" : {
        "operationId" : "featureChange_AeronauticSrf_items",
        "bindings" : {
          "qos" : 0,
          "retain" : false,
          "bindingVersion" : "0.1.0"
        },
        "message" : {
          "$ref" : "#/components/messages/featureChange_AeronauticSrf"
        }
      },
      "servers" : [ "t19" ]
    },
  }
}
```

The deployment uses the broker for subscribing to the messages from the testbed. The ldproxy instance offers access to approximately 30 different feature collections. Whenever something is

created or changed in a feature collection, a message is sent to the broker. A screenshot showing the channels being monitored through an application called MQTT Explorer is presented in Figure 23.

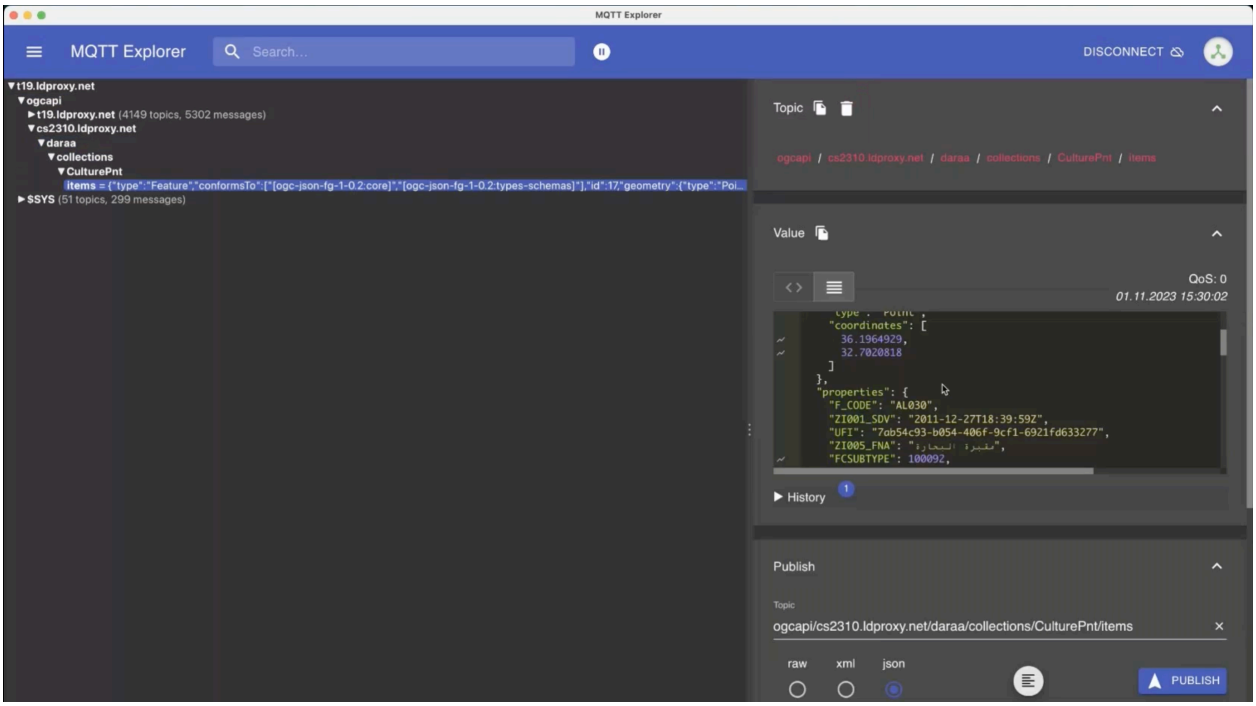


Figure 23 — ldproxy accessed from MQTT Explorer

4.3.2.4. Securing the Daraa API

The Daraa API in the cs2310.ldproxy.net deployment also supported operations to create, replace, update, or delete features in any of the feature collections. Such operations will in almost all cases be restricted to authorized users.

The deployment was therefore connected to an Identity Provider supporting OpenID Connect. This supported:

- a user “writer” to log-in in the Swagger UI client of the OpenAPI definition using the OAuth2 Authorization Code Flow and execute create, replace, update, or delete operations; and
- a client “writer” to execute create, replace, update, or delete operations using the OAuth2 Client Credentials Flow using a client secret.

Both capabilities and flows were demonstrated during the sprint. All changes to features could be monitored via the PubSub broker described above.

4.3.3. OSGeo pycsw

The pycsw project upgraded support for the OGC API — Records `q` parameter based on the latest updates (changes in Pull Request 912 and Pull Request 913). In addition, record parsing and ingest updates and fixes were implemented, as described in Pull Request 914, given specification / schema updates.

4.3.4. Safe Software FME

The purpose for Safe Software's participation in the code sprint was to run some experiments with a logical model based on MUDDI in order to encode underground data into GML and other popular encodings, such as GeoJSON, GeoPackage, and ESRI Geodatabase. Safe Software also ran some transformations from CityGML2 and IFC to CityGML3. To accomplish these data transformations, the team from Safe Software used the FME data modeling platform. FME is a geospatial data integration and automation platform that supports extract, transformation, load, and many other data and schema transformation and geoprocessing functions. It also supports data conversion between more than 500 different data formats, services, and APIs. The 2021 version of FME Desktop is Certified OGC Compliant to GML 3.2.1, GeoPackage 1.2, KML 2.2 and GeoTIFF (https://www.ogc.org/resources/product-details/?pid=1681).

4.3.4.1. MUDDI

In order to exercise the MUDDI GML encoding for the code sprint, Safe Software obtained Sanitary system data from the City of Burnaby in Canada from their publicly accessible data portal. The data is available from the City as open data. The data includes information about the Sanitary system and connectors, including the main service lines and service connections, as well as manhole covers. Using the FME product, the team sought to load this Burnaby utilities data into a schema, based on the logical model described by the MUDDI application schema, that had been provided to sprint participants in support of the code sprint.

The workflow for converting from plain GeoJSON to MUDDI GML is illustrated in Figure 24. The workflow takes source data and copies attributes across to output feature types. Excess attribution that is not required in the output feature type can be thrown away. A Network feature acts as a parent to all of the other associated features. Therefore, the mains and service lines are associated with a network. This includes fittings, mains and service lines to which some date fields are added. The workflow also creates a service area which represents the bounding box around the features.

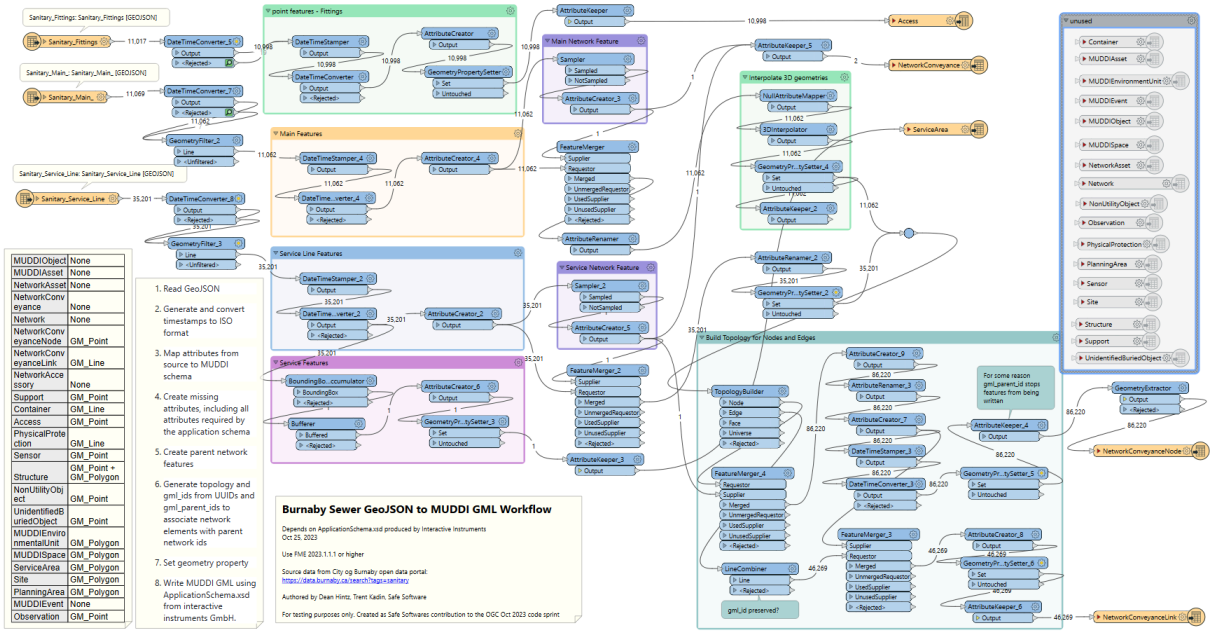


Figure 24 — Screenshot of FME workflow for converting Burnaby underground data from plain GeoJSON to MUDDI GML

A summary of the workflow is below:

1. read GeoJSON;
2. generate and convert timestamps to ISO format;
3. map attributes from source to MUDDI schema;
4. create missing attributes, including all attributes required by the application schema;
5. create parent network features;
6. generate gml_ids from UUIDs and gml_parent_ids to associate network elements with parent networks;
7. set geometry property; and
8. write MUDDI GML using ApplicationSchema.xsd from interactive instruments GmbH.

The workflow invokes the GML writer capability of FME. An overview of the GML writer's schema capability is available on the Safe Software website (https://community.safe.com/s/article/gml-writing-with-application-schemas).

The team from Safe Software reported encountering issues whereby some of the attribution required by the MUDDI schema was not available from the source dataset. Fortunately, in such situations, the software logged detailed error messages that listed the required attributes that had not been filled and the location in the output data where the errors occur. There were also

some errors encountered where invalid default values were set — such as wrong date format, wrong data type (string vs float), or invalid coded domain value. The participants were then able to add the required attributes using an AttributeCreator function and set these to valid default values. A NullAttributeMapper function was used to set default values where null values are not allowed. DateTimeConverter function was used to convert date time values to the valid ISO format. After this the XML schema validation was successful.

Based on the above tests, the FME platform was able to read the publicly available Burnaby GeoJSON data, write it to MUDDI GML, and then read it back, as illustrated in Figure 25. From the screenshot it can be seen on the top left that there were 11,000 fittings. The screenshot also shows the two networks. Those networks are associated with service lines. One of the service lines is associated with names. The Network conveyance link is the actual line. The Service Area is represented by a bounding box.

While this test did not utilize all the available MUDDI feature types, it did accomplish the following:

1. wrote typical simple MUDDI geometry types including null, point, line, polygon / surface;
2. wrote georeferenced data with the defined OGC WGS84 CRS (urn:ogc:def:crs:EPSG::4326);
3. mapped available properties to the MUDDI schema;
4. created populated xlink:href links to associate network edges (NetworkConveyanceLinks) and nodes (NetworkConveyanceNodes);
5. wrote both 2D and 3D geometries, in some cases within the same feature type (3D geometries were derived and interpolated for NetworkConveyanceLink from the start and end depth properties where available using the FME 3DInterpolator function); and
6. successfully validated the output GML against the interactive instruments MUDDI application schema.

Note that the schema was only made available to sprint participants less than a week before the code sprint. It was also the first draft schema available that had any geometry definitions. Also it is worth noting that while NetworkConveyanceNode was designed to support point features, for some reason it was not possible to successfully generate point geometries for this feature type. This may be the result of a problem with the underlying draft GML application schema. As a workaround the team from Safe Software wrote the point geometry as a GeoJSON blob into the description field.

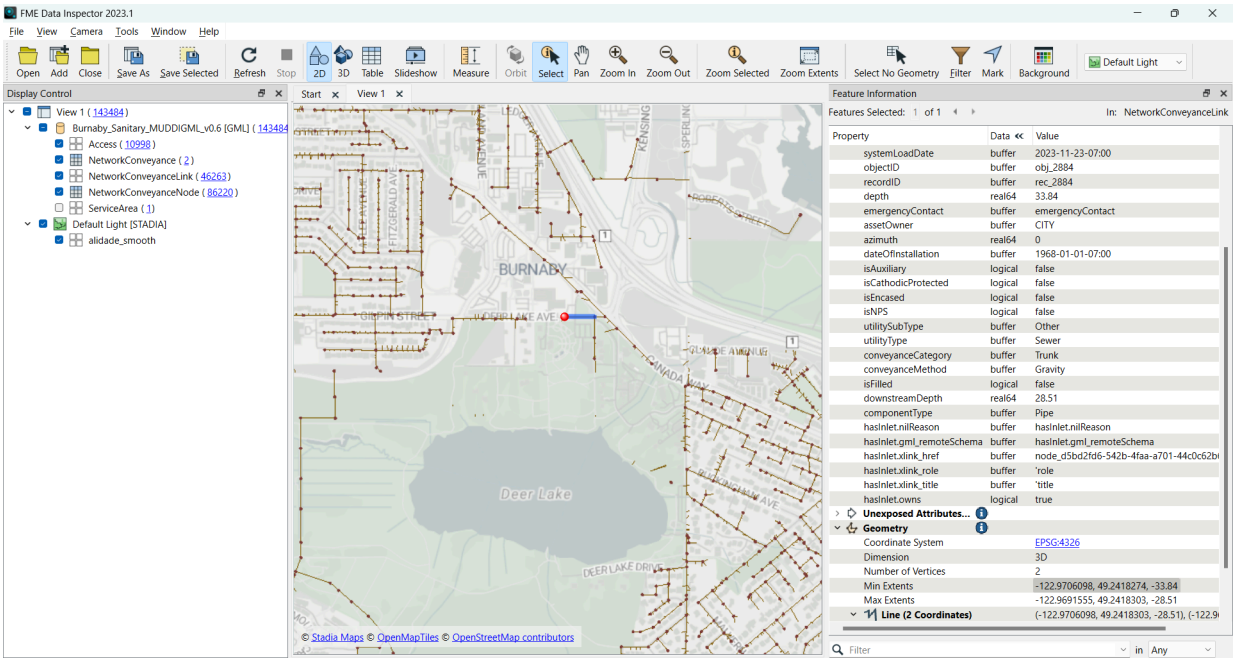


Figure 25 — Screenshot of MUDDI GML data displayed in FME for a NetworkConveryanceLink feature. Note the reference to the associated node xlink_href. Also note the 3D geometry with depth stored as z values on the feature coordinates.

On the right-hand side of Figure 25 is shown the attribution from the data. Most of the attribution is carried through from the source data, however for illustrative purposes in this Code Sprint some additional synthetic attribution was inserted. The workflow also added an elevation dimension to the geometries, which was taken from another feature which had up and down elevation feature properties.

During code sprint discussions, it was noted that some applications may have trouble consuming data in the GML encoding. To make the MUDDI test results more widely sharable, the Safe Software team decided to also convert MUDDI GML to a number of other common encodings, including GeoJSON, OGC GeoPackage, and ESRI Geodatabase. This involved a fairly simple MUDDI GML to GeoJSON and MUDDI GML to database conversion workflow. In some cases limitations on destination schemas required conversion of dot notation to slash, such as for Geodatabase. The team from Safe Software also read the GeoJSON point value from the NetworkConveyanceNode description field so that in these datasets NetworkConveyanceNode has a valid point geometry. It was noted that overall the most compact output form seemed to be GeoJSON as the optional fields with missing values were simply not written out, resulting in a more streamlined output.

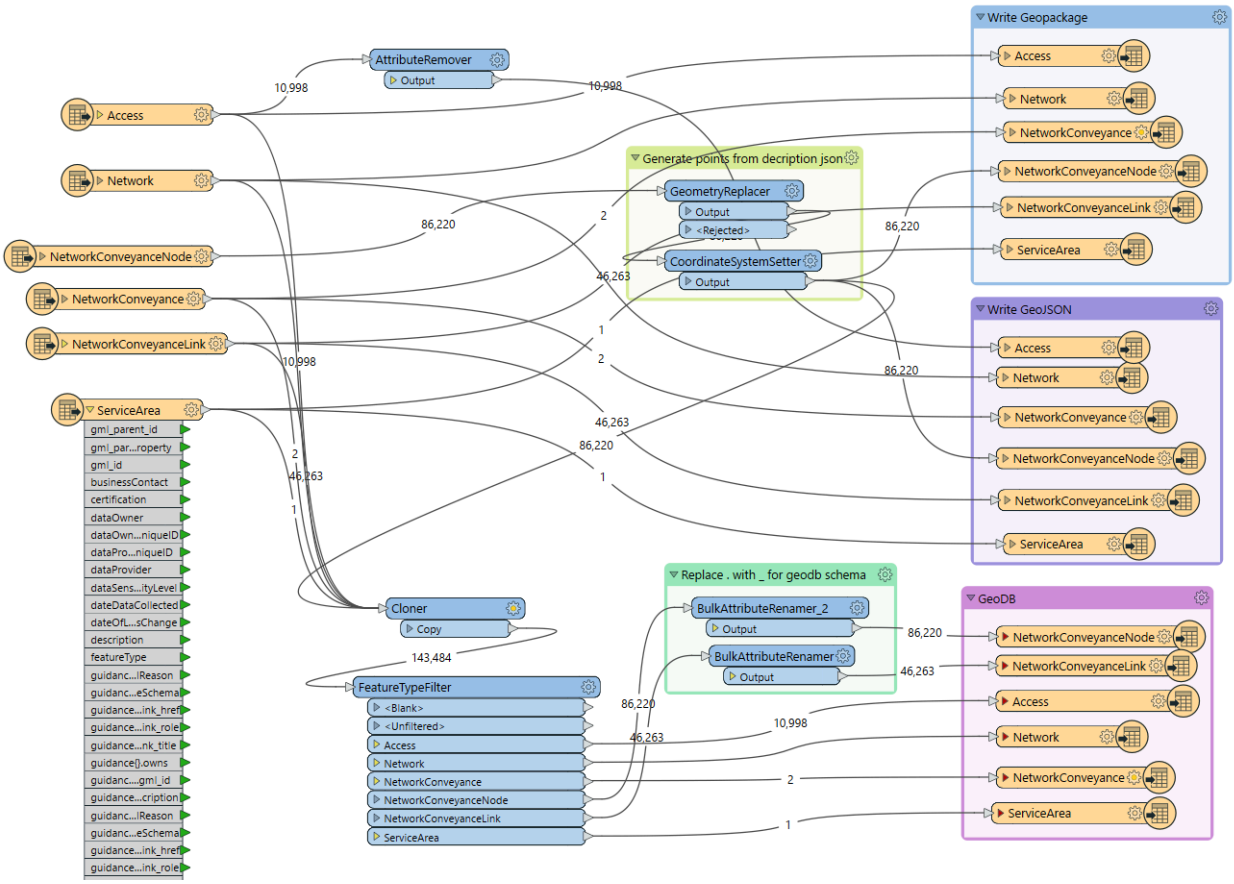


Figure 26 — Screenshot of FME workflow for converting from MUDDI GML to MUDDI compliant GeoJSON, GeoPackage and Geodatabase.

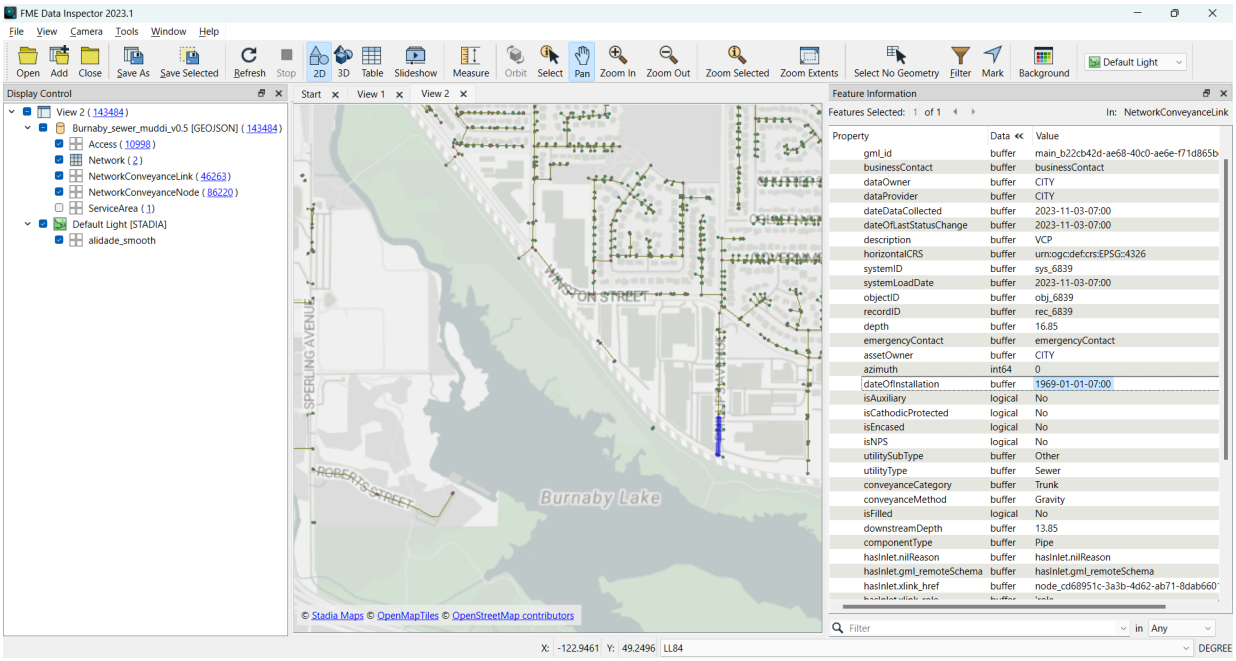


Figure 27 — Screenshot of MUDDI GeoJSON data displayed in FME. Note the simplified schema with empty properties not shown.

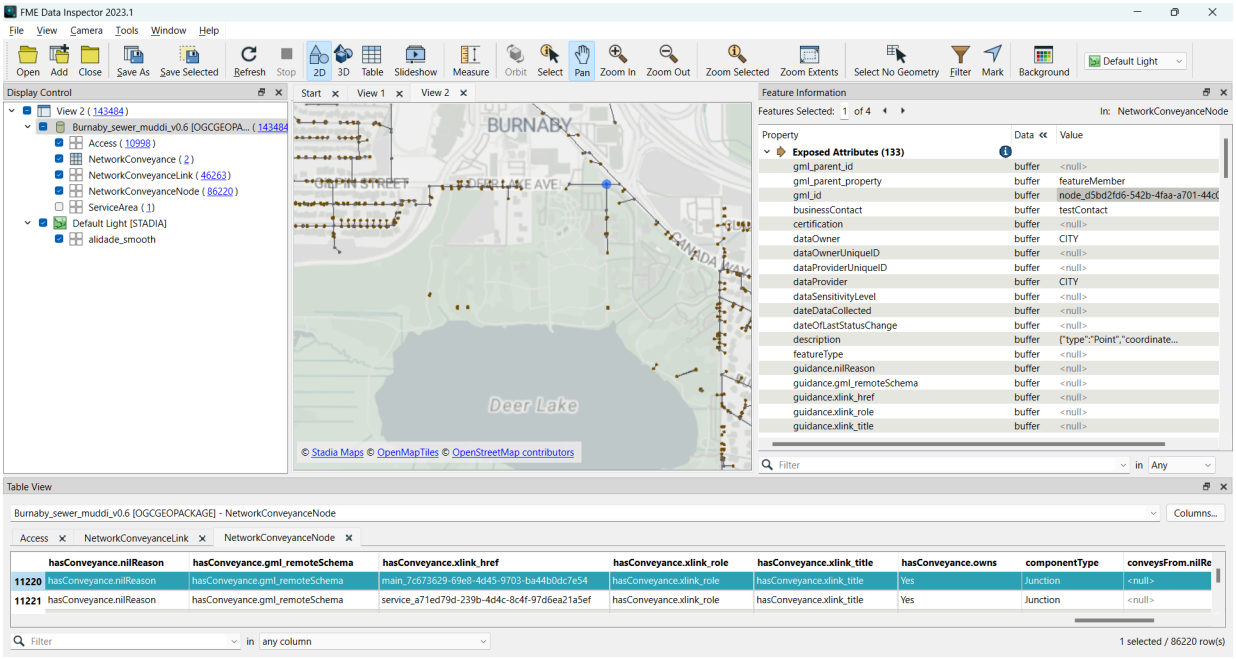


Figure 28 — Screenshot of MUDDI GeoPackage data displayed in FME showing a NetworkConveyanceNode feature. Note that the gml_id of this node matches the hasInlet.xlink_href of the connected NetworkConveyanceLink feature shown in the MUDDI GML output above.

4.3.4.2. CityGML

Sprint participants from Safe Software also worked on a demonstration of CityGML2-to-CityGML3 conversion of Montreal data. All attribution (apart from appearances) was pulled across. An illustration of the workflow for converting from CityGML2 to CityGML3 is shown in Figure 29.

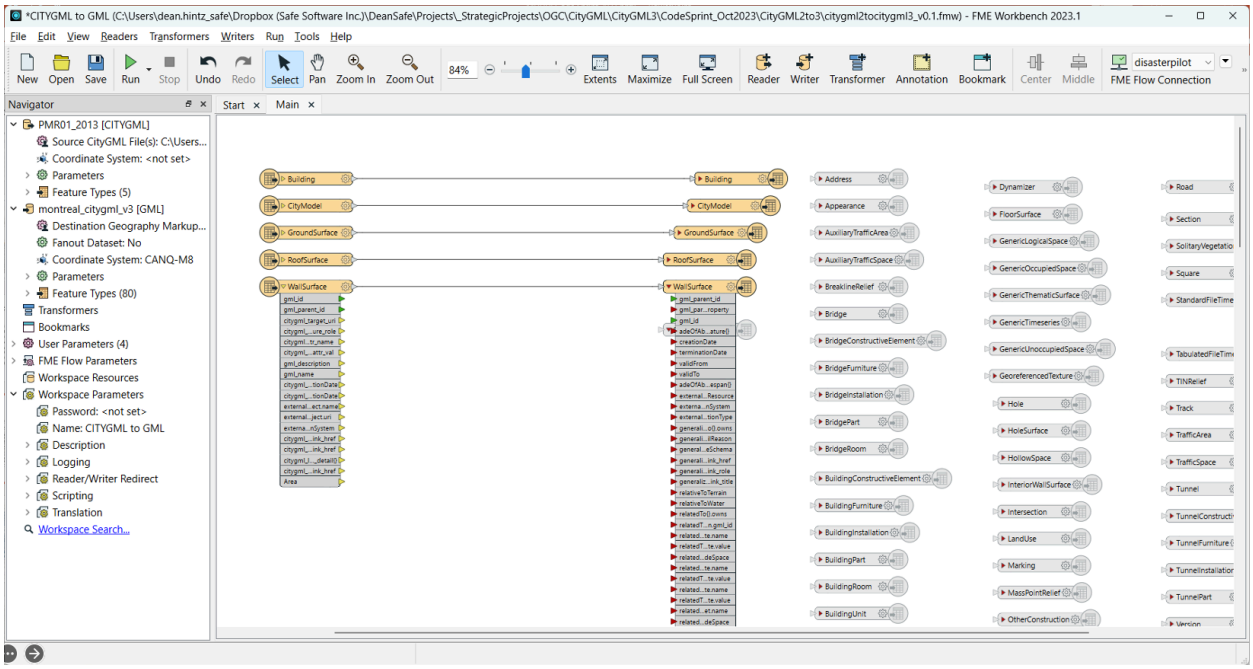


Figure 29 — Screenshot of workflow for converting from CityGML2 to CityGML3

A separate workflow was created for converting from IFC to CityGML3. An illustration of the workflow is shown in Figure 30.

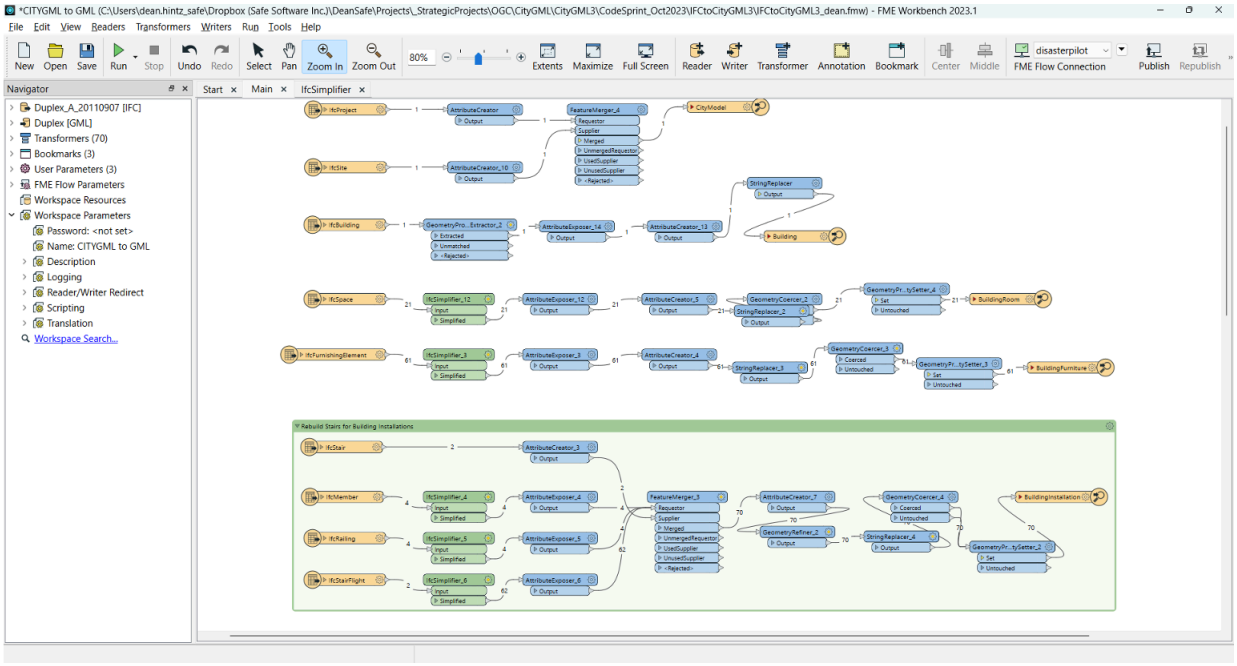


Figure 30 — Screenshot of workflow for converting from IFC to CityGML3

The output CityGML is presented in Figure 31. The figure shows rooms in a building with one of the rooms selected. The attributes of the selected room, which are presented on the right-hand side, report that the room is a living room.

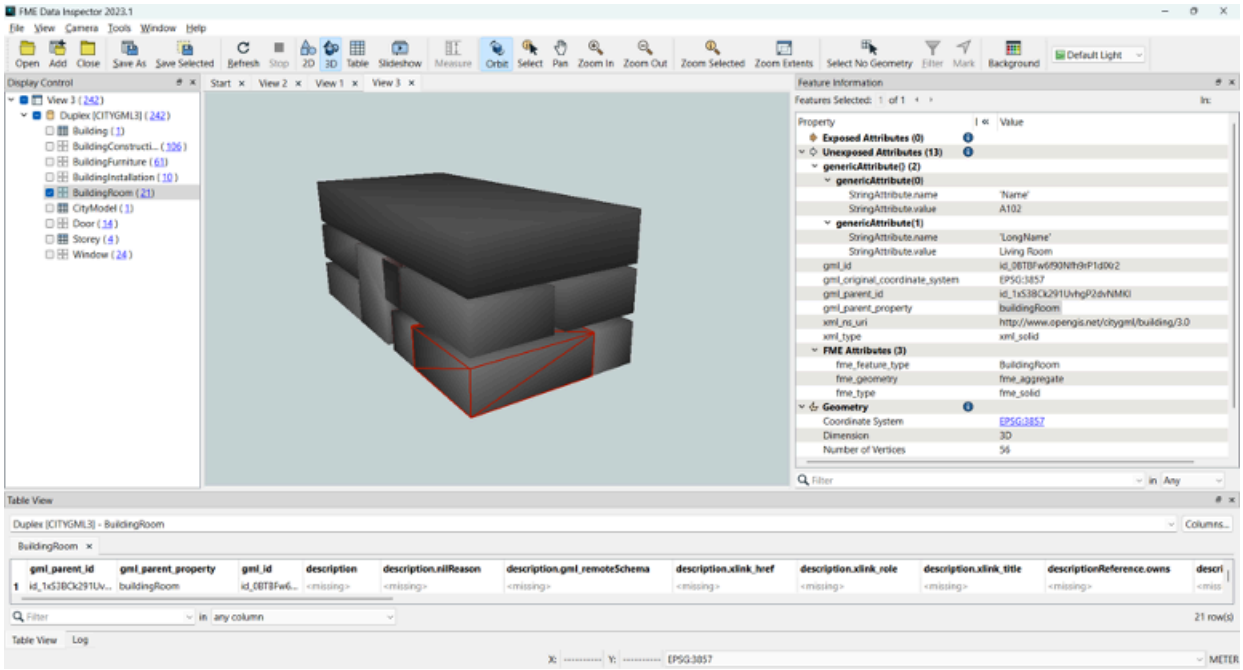


Figure 31 — Screenshot of the output CityGML data displayed in FME

4.3.4.3. Summary

In summary, sprint participants from Safe Software were able to use the FME platform to write schema valid MUDDI GML using the application schemas provided by interactive instruments. The workflows for writing to MUDDI GML captured multiple two-dimensional (2D) and three-dimensional (3D) geometry types (points, lines, polygons). Topology was also populated with xlink_hrefs. The team was also able to convert this MUDDI GML into several other common encodings including OGC GeoPackage, ESRI Geodatabase, and GeoJSON.

The sprint participants from Safe Software also started experimenting with the CityGML3 Dynamizer to explore how it might work in future versions of the workflow. They successfully converted CityGML2 data to CityGML3 data, as well as IFC data to CityGML3 data. The participants from Safe Software were able to store IFC spaces as rooms and support a range of other new CityGML3 feature types and properties and also confirmed that the output back with the new FME CityGML3 reader could be read.

For more information, examples, and sample datasets produced by Safe Software for this code sprint and Safe Software's other OGC activities please see:

1. Open-Standards-Code-Sprint/muddi/safe-software
2. Safe Software participation in OGC Code Sprints
3. FME and OGC Open Standards

4.3.5. Hexagon Catalog Explorer

Sprint participants from Hexagon demonstrated the implementation of a client application for OGC API — Records and demonstrated that the client application could access 3rd party catalogs served through implementations of OGC API — Records. One of the issues that was sought to be resolved during the code sprint was the format and source of the thumbnails. For example, whether the thumbnails could be served by a WMS or OGC API — Maps implementation. Following the code sprint, the issue was discussed by the SWG and consensus reached.

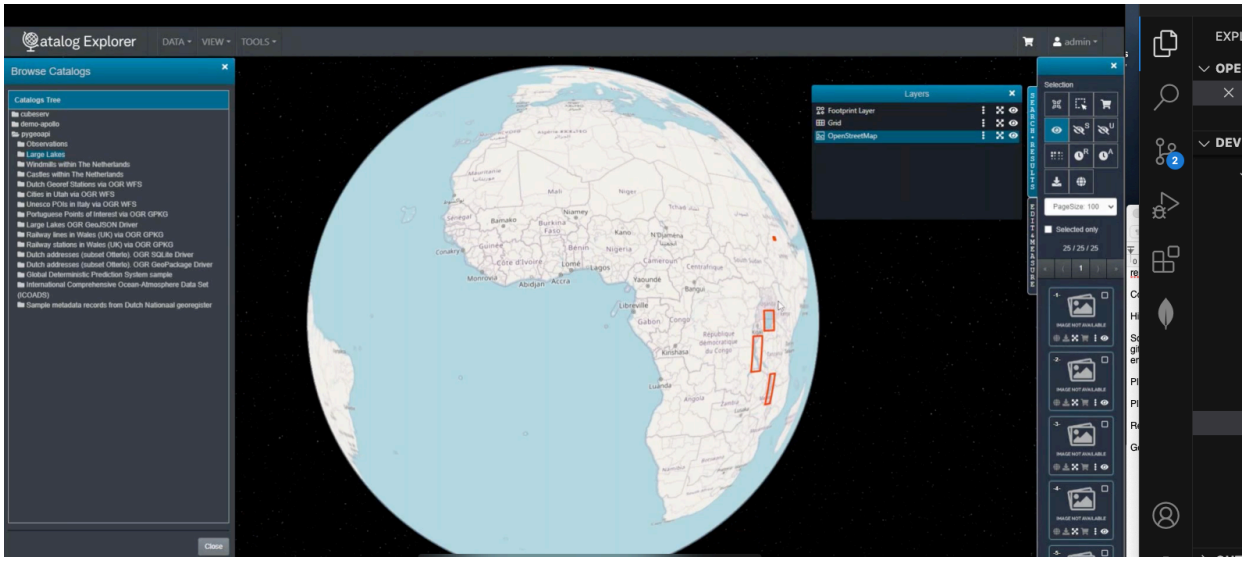


Figure 32 — Screenshot of the Hexagon Catalog Explorer accessing an implementation of OGC API - Records

4.3.6. Geonovum testing of JSON-FG support in GDAL

Sprint participants from Geonovum worked on testing the JSON-FG candidate Standard and evaluating GDAL's implementation of the candidate Standard. To facilitate the evaluation, a series of test scenarios were specified that include tests of transformation of GML to JSON-FG, GeoPackage to JSON-FG, and transformation from 3D data formats to JSON-FG. A more detailed discussion of the tests and their results is presented in an associated report [20]. Although the tests did not reveal any bugs or issues found in GDAL's support of JSON-FG, there were a number of findings that could inform future versions of GDAL.

The tests were applied to source data that contained features that were referenced to the Dutch coordinate reference system Rijksdriehoeks (RD). During transformation of feature data to JSON-FG, by default GDAL creates a geometry member that has coordinates that are referenced to WGS84. Further, also by default, GDAL creates a place member with coordinates obtained from the source data during transformation of feature data to JSON-FG. Although this conforms with the JSON-FG candidate Standard, it may not be what some users expect.

One of the findings was that when GDAL cannot identify a source property for the temporal information, it fills the time member of the JSON-FG file with a null value. There does not appear to be a way to point GDAL at an appropriate source for filling the time member. However, as GDAL looks for an element called `time`, `time_start`, or `time_end`, it is possible to use a GDAL tool called ogr2ogr to remap the properties of the source data to an output property that has a name that GDAL expects (e.g., time). Another finding was that there is currently no apparent option to have GDAL create a JSON-FG file that references a JSON Schema file that describes the featureSchema.

4.3.7. OSGeo pygeoapi

4.3.7.1. Implementation of a CSW Facade

The pygeoapi project implemented an OGC API — Records CSW Façade plugin as a means to bridge CSW implementations. Note that support is bound to CSW 2/3 and the CSW Record core model (ISO 19115/19139 support can be implemented as future work). This work was merged to the core pygeoapi codebase.

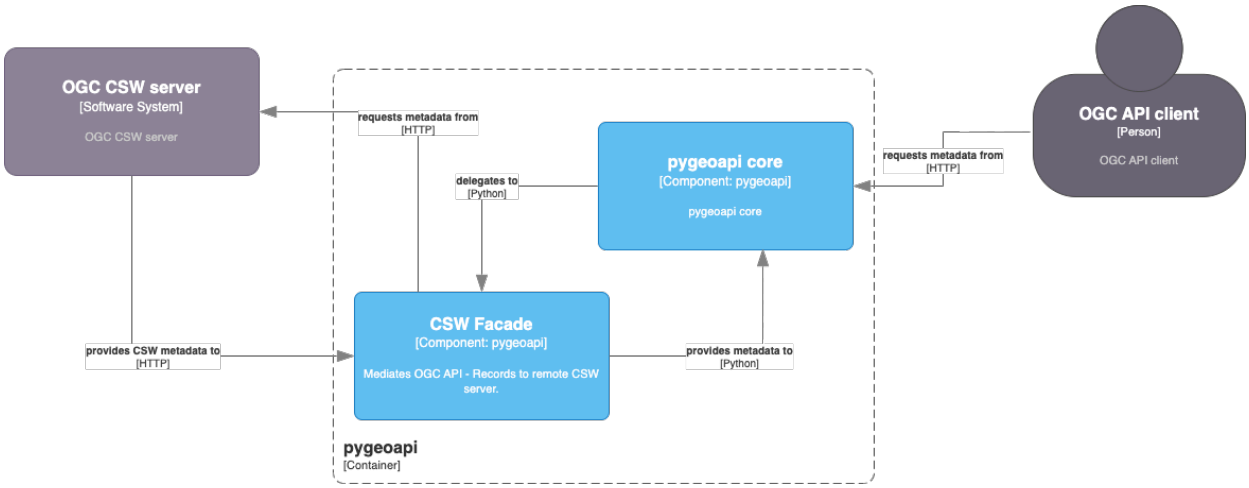


Figure 33 — Screenshot of the pygeoapi OGC API - Records provider architecture for the CSW Facade.

A live demonstration was provided using the WMO World Ozone and Ultraviolet Radiation Data Centre (WOUDC) CSW server.

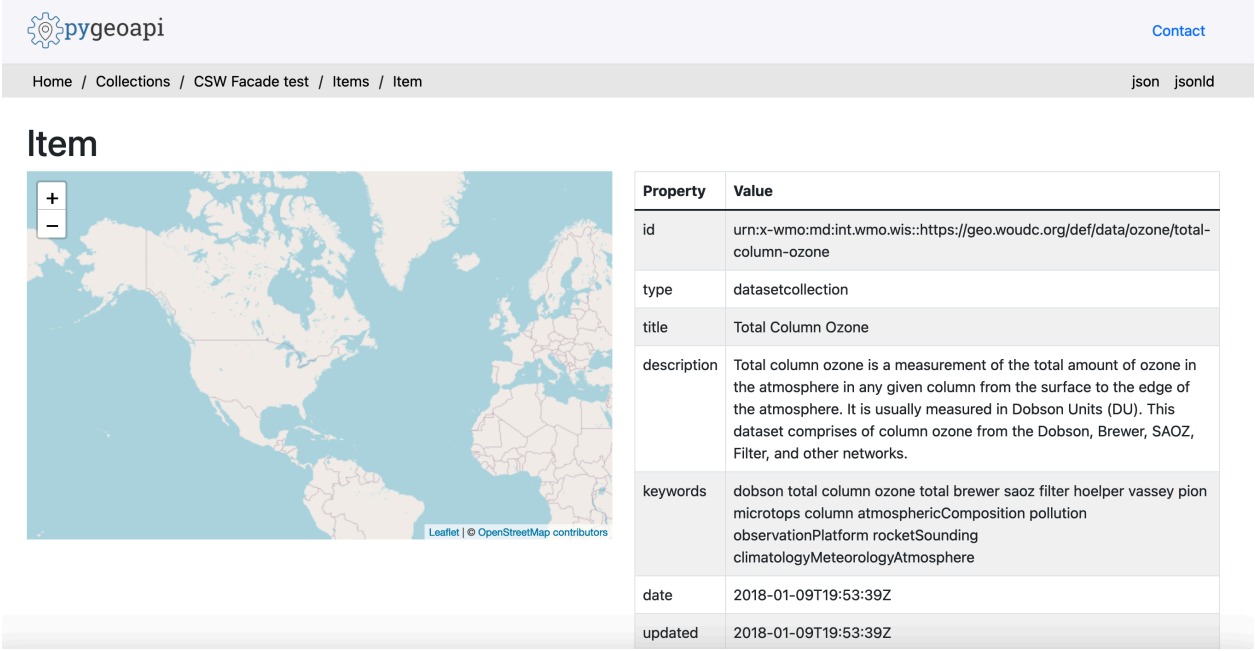


Figure 34 — Screenshot of the pygeoapi OGC API - Records provider for the CSW Facade.

4.3.7.2. Support for Publish/Subscribe operations

A live demo was given which exemplified the capabilities of PubSub as a workflow for metadata catalog management, using the WMO WIS2 Global Discovery Catalogue Reference Implementation (powered by pygeoapi). In this demo, the catalog is subscribed to an MQTT broker which is publishing messages of new discovery metadata records. On notification of new metadata, the catalog is triggered to validate and ingest the metadata into a metadata collection, thereby making it available via OGC API — Records.

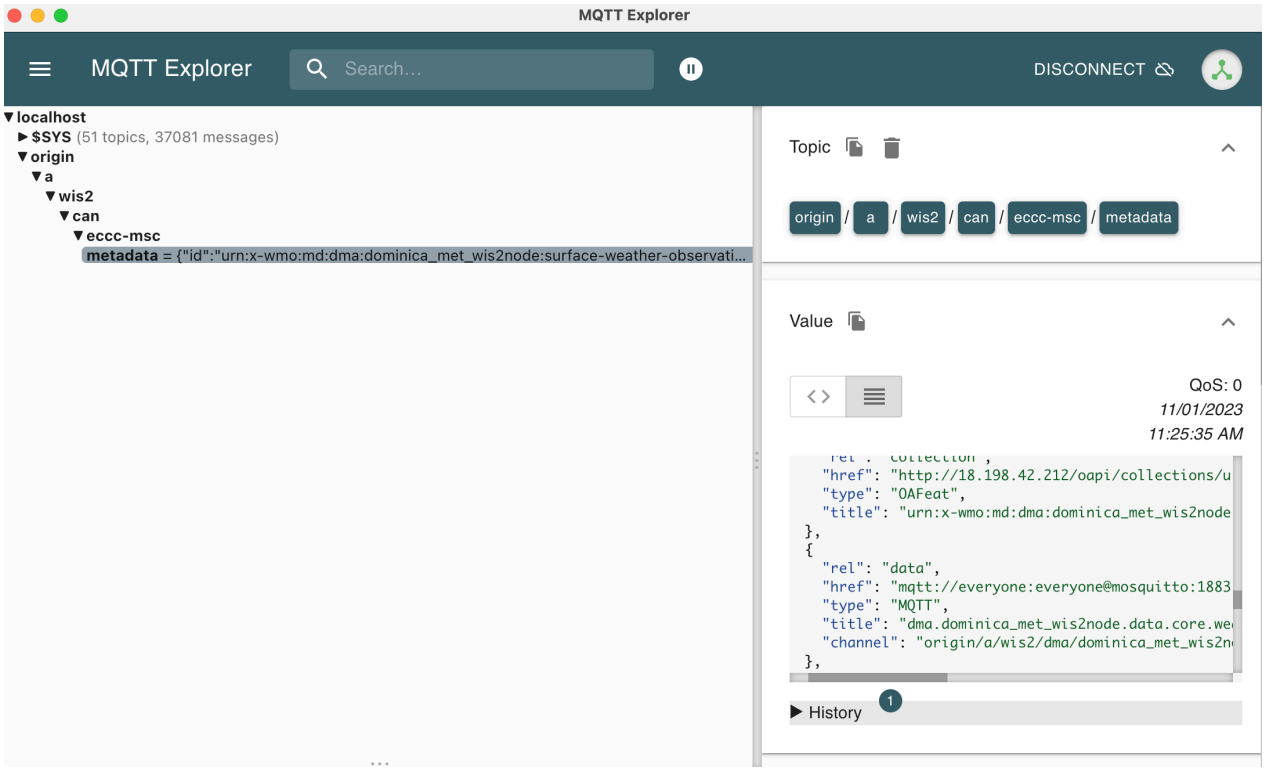


Figure 35 — Screenshot of MQTT Explorer with metadata notification updates.

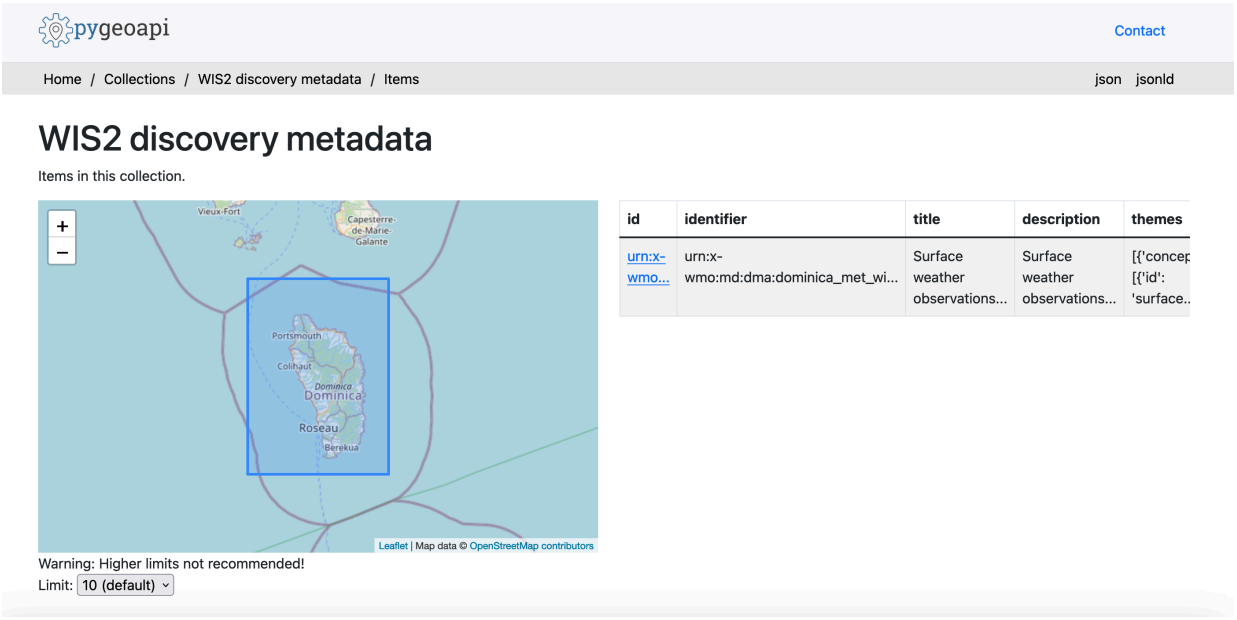


Figure 36 — Screenshot of metadata published to OGC API - Records using pygeoapi.

4.4. Products applied to OGC Standards and GIMI in this code sprint

4.4.1. pygeoapi support for the draft GIMI profile

Sprint participants from Geobeyond worked on the GIMI thread of the code sprint with the goal to implement the ability to serve files using OGC API – Coverages and OGC API – Tiles capabilities offered by pygeoapi. An additional goal was to enable a workflow that supports modification of the files. The libraries that were modified or implemented for this work are available on the gimirrai repository on GitHub.

As an initial objective, the sprint participants set out to see whether the libheif library, which is commonly used to read HEIF files, could be enabled to support reading and writing of GIMI files. libheif is used by GDAL to provide support for HEIF, so the expectation was that it would be possible to use libheif to leverage GDAL's HEIF support via rasterio. However, upon testing libheif, it was found that Python does not support some of the needed capabilities specific to the GIMI profile. This lack of support in libheif means that popular raster-handling tools based on GDAL are not by default able to support GIMI files. Some modifications are thus required to enable such tools to support for the format.

With help from Brad Hards, the sprint participants compiled a patched version of libheif that could read GIMI files and then compiled GDAL, pillow-heif and rasterio to enable the reading of GIMI files through Python, which was successful. Note that, even though the spring participants were now working with a version of libheif that was able to read the sample GIMI files, and even though GDAL was now able to leverage libheif to provide read support, GDAL still did not know how to read the sample GIMI files' georeferencing metadata, as this is stored in a KLV-encoded bytestream, which is specific to GIMI.

The sprint participants were able to decode and parse the GIMI KLV bytestream by means of reading the GIMI files with pillow-heif and then implementing a custom KLV bytestream parser. Although it did work OK, the implementation was fairly low level, relying on knowledge of the MISB ST0601 standard and on decoding bytes to their corresponding types. Decoding KLV enabled them to review the geographic bounds declared in the metadata of the GIMI files. Figure 37 shows the geographic bounds of a sample GIMI file displayed on a pygeoapi interface.

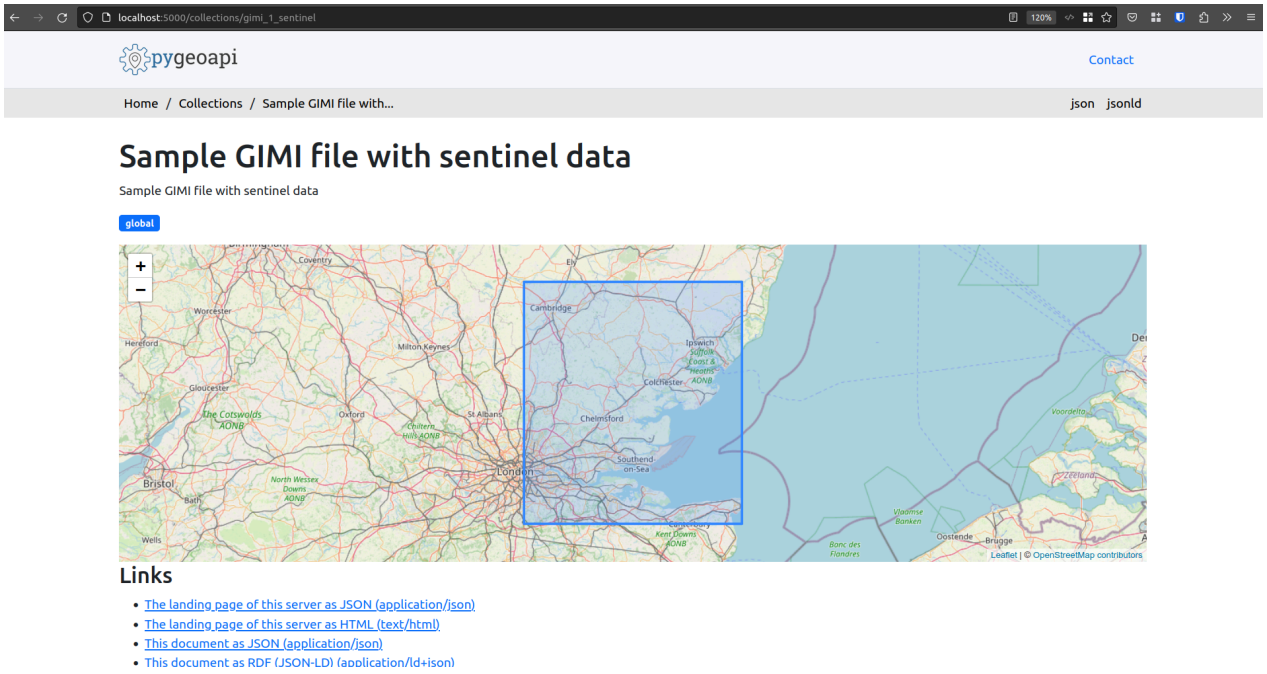


Figure 37 — Screenshot of pygeoapi's HTML interface, showing a GIMI coverage collection. The collection's geographic bounds are displayed by pygeoapi.

Afterwards, the sprint participants sought to enable pygeoapi to support the reading of GIMI files, which would make it possible to serve the imagery files through interfaces conforming to OGC API Standards.

The pygeoapi integration efforts relied upon pygeoapi's existing extensibility mechanism, whereby it is possible to write custom providers and plug them into dataset collections via configuration.

For OGC API — Coverages, the sprint participants were able to write a custom provider inspired by the stock rasterio provider. The GIMI provider differed in the sense that instead of letting rasterio retrieve the geographical bounds transparently, via GDAL, it needed to use the KLV bytestream parser mentioned above.

For OGC API — Tiles, implementation was a bit more involved. In order to leverage existing Python-based libraries (which are usually using GDAL bindings) to generate tiles, GDAL would need to be able to properly recognize a GIMI file's georeferencing metadata. The team was able to put together a solution that relied upon the GDAL Virtual Format (VRT), together with the already mentioned custom KLV parser. The VRT approach then unlocked the usage of the rio-tiler library, which is a rasterio plugin that provides means to generate raster tiles. The final implementation was a pygeoapi provider that tied all parts together, reading KLV metadata, then using a GDAL VRT definition to allow rio-tiler to be able to read the sample GIMI files and generate tiles. As such, and for brevity, the implemented provider was simplified to:

- serve tiles, the other OGC API related endpoints were not given much attention;
- support only the familiar WebMercatorQuad tile matrix set;
- serve tiles in PNG format; and

- generate tiles at runtime, without any caching strategy.

The end result was then visualized on a simple leaflet-based web client, which used the pygeoapi endpoints to serve tiles for sample GIMI files. Figure 38 shows a screenshot of a GIMI file displayed on a web map served by pygeoapi.

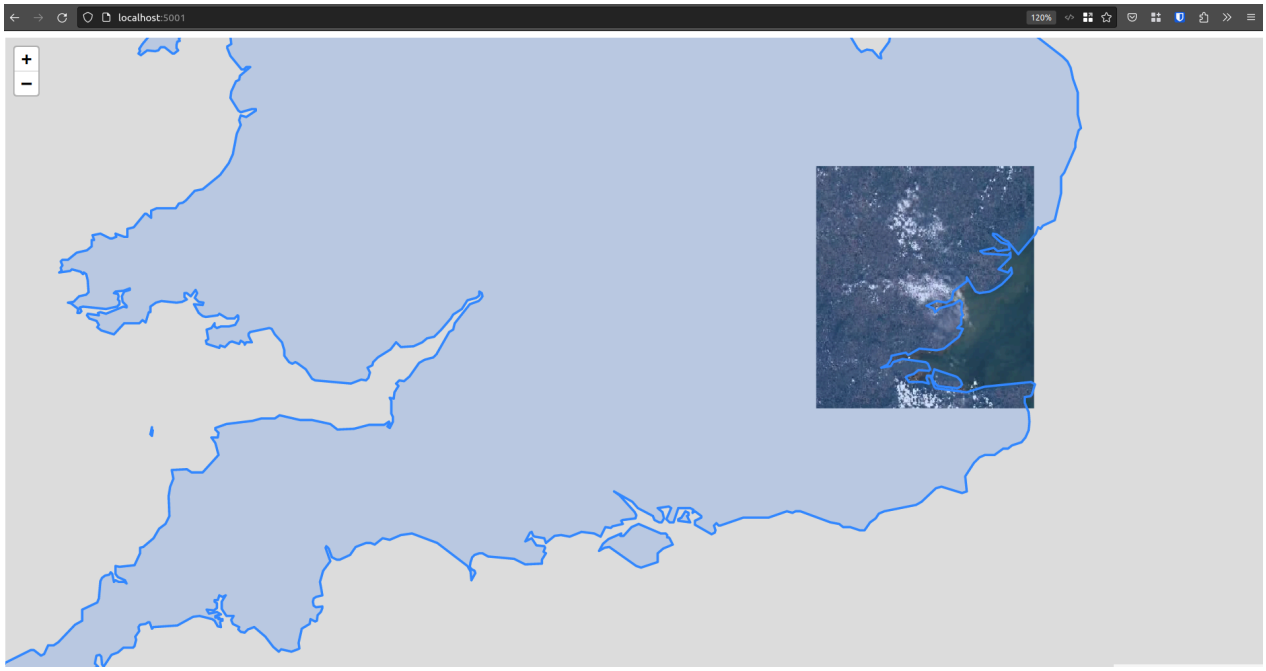


Figure 38 — Screenshot of the leaflet client consuming tiles, which are served by pygeoapi, in the PNG format, and are generated from the GIMI file.

The final goal was implementing support for CRUD operations on GIMI files via a Web API, leveraging OGC API Standards. Unfortunately, this goal proved unattainable during the code sprint, as the team found substantial technical blockers that could not be solved during the code sprint.

Initial research showed that GDAL does not have support for creating HEIF files. So the plan was to use libheif instead, by leveraging its existing Python bindings. However, none of the efforts related to using pillow-heif to create or update one of the sample GIMI files was successful. Several attempts were made, by leveraging the different Pillow and pillow-heif capabilities, but all attempts ended abruptly due to various lower-level errors. Having failed at being able to modify an existing GIMI file with the existing libraries, the team could not move forward with the web and pygeoapi related aspects of the goal.

In conclusion, the team found that it is currently not very easy to work with GIMI files at the Python level (i.e. those conforming to draft version 0.4 of the GIMI profile). Reading and georeferencing GIMI files is possible, but requires a larger effort, especially when compared to common geospatial file formats. Creating and updating existing GIMI files seems not to be possible through the more popular tools used in the Python ecosystem (e.g., Pillow). Therefore, a custom writer is required to support the creation and updating of GIMI files.

On a more positive note, because Python tools mostly depend on libheif and GDAL, if proper GIMI support would be added to these lower-level libraries, working with GIMI files would

become as easy as working with GeoTiff and other popular formats. Having this lower-level support in place would open up the possibility of using rasterio, rio-tiler, and other libraries with GIMI files in a transparent way. Additionally, pygeoapi proved to be a very flexible choice when implementing OGC API-based servers, as it is easy to integrate custom bits of code.

4.4.2. Storage of GIMI files through a Web API

Ahead of the three-day code sprint, one of the objectives was to demonstrate the update and storage of GIMI files through the Create, Replace, Update, Delete (CRUD) approach adopted by other OGC API Standards. However due to limitations in the built-in capabilities of the Python ecosystem, it was not possible to implement the approach during the three-day code sprint. The experimentation conducted during the code sprint also highlighted that the OGC API — Processes Standard may be more appropriate for supporting the storage and update requirements. This is because OGC API — Processes allows for different types of payload to be submitted to an API. Therefore, following the three-day code sprint, OGC staff prototyped an approach that leverages the OGC API — Processes Standard to achieve storage of GIMI files.

The prototype involved modifying the hello-world process that comes built into pygeoapi following the instructions on the OGC e-learning resource. The process implemented by the prototype was called 'imagery-utils' and was designed to receive a GIMI image encoded as a base64 string and then to store that image locally on the server.

To create a process in pygeoapi for storing an image through the OGC API — Processes interface, the first step is to clone and build pygeoapi by following the steps below. The steps are copied from the pygeoapi.io website.

```
python3 -m venv pygeoapi
cd pygeoapi
. bin/activate
git clone https://github.com/geopython/pygeoapi.git
cd pygeoapi
pip3 install -r requirements.txt
python3 setup.py install
cp pygeoapi-config.yml example-config.yml
vi example-config.yml  # edit as required
export PYGEOAPI_CONFIG=example-config.yml
export PYGEOAPI_OPENAPI=example-openapi.yml
pygeoapi openapi generate $PYGEOAPI_CONFIG --output-file $PYGEOAPI_OPENAPI
pygeoapi serve
# in another terminal
curl http://localhost:5000  # or open in a web browser
```

The next step is to modify the example-config.yml file by adding the following process to the resources element.

```
    imagery-utils:
        type: process
        processor:
            name: ImageryUtilities
```

The next step is to add the following paths to the example-openapi.yml.

```
  /processes/imagery-utils:
    get:
      description: An example imagery utility process.
      operationId: describeimagery-utilsProcess
```

```
      parameters:
      - $ref: '#/components/parameters/f'
      responses:
        '200':
          $ref: '#/components/responses/200'
        default:
          $ref: '#/components/responses/default'
      summary: Get process metadata
      tags:
      - imagery-utils
  /processes/imagery-utils/execution:
    post:
      description: Execution of an example imagery utility process.
      operationId: executeimagery-utilsJob
      requestBody:
        content:
          application/json:
            example:
              inputs:
                message: An optional message.
                name: World
            schema:
              $ref: https://schemas.opengis.net/ogcapi/processes/part1/1.0/
openapi/schemas/execute.yaml
        description: Mandatory execute request JSON
        required: true
      responses:
        '200':
          $ref: '#/components/responses/200'
        '201':
          $ref: https://schemas.opengis.net/ogcapi/processes/part1/1.0/openapi/
responses/ExecuteAsync.yaml
        '404':
          $ref: https://schemas.opengis.net/ogcapi/processes/part1/1.0/openapi/
responses/NotFound.yaml
        '500':
          $ref: https://schemas.opengis.net/ogcapi/processes/part1/1.0/openapi/
responses/ServerError.yaml
        default:
          $ref: '#/components/responses/default'
      summary: Imagery utility process execution
      tags:
      - imagery-utils
```

Then add the following to the root-level list of tags in the example-openapi.yml file.

```
- description: Imagery processing utilities
  name: imagery-utils
```

Thereafter, add the ImageryUtilities process to the plugin.py file as shown below.

```
    'process': {
        'ImageryUtilities': 'pygeoapi.process.imagery_utilities.
ImageryUtilitiesProcessor'
    },
```

Thereafter copy and modify the hello_world.py file to create one called imagery_utilities.py. Within that file replace 'hello-world' with 'imagery-utils,' 'HelloWorld' with 'ImageryUtilities,' and 'hello_world' with 'imagery_utilities.' Most importantly the execute function should look as shown below.

```
import base64, binascii,time
from urllib.request import urlopen
```

```
    def execute(self, data):

        mimetype = 'application/json'
        image_dataset = data.get('message', None)

        url_of_results = 'No results'

        output_filename = "savedimage"+str(time.time_ns())
        print(output_filename)

        fh2 = open("../storage/"+output_filename+".txt", "w")
        print(image_dataset)
        fh2.write(image_dataset)
        fh2.close()

        try:
            image = base64.b64decode(image_dataset, validate=True)
            with open("../storage/"+output_filename+".heic", "wb") as f:
                f.write(image)
                url_of_results = 'Storage successful'
        except binascii.Error as e:
            print(e)


        if image_dataset is None:
            raise ProcessorExecuteError('Cannot process')


        value = ' {}'.format(url_of_results).strip()


        outputs = {
            'id': 'result',
            'value': value
        }

        return mimetype, outputs
```

Thereafter, to start the application, run the following commands.

```
python3 setup.py install
pygeoapi serve
# in another terminal
curl http://localhost:5000  # or open in a web browser
```

To test the process, Postman, curl, or any other HTTP client can be used. Postman is an application that offers a Graphical User Interface for testing HTTP endpoints such as those provided by Web APIs. In contrast, curl offers some of the same capability but through a command line interface.

Now from an HTTP client such as Postman or curl, send the following request via POST to http://localhost:5000/processes/imagery-utils/execution

```
{
        "inputs": {
            "name": "example",
            "message": "AAAAJGZ0eXBoZWljAAAAAG1pZjFoZWljZ2VvMXVuaWZtaWFmAAAFr21
ldGEAAAAAAAAAIWhkbHIAAAAAAAAAAHBpY3QAAAAAAAAAAAAAAAAAAAAAAAAADnBpdG0AAAAAAIAAAI1aW
RhdDw/eG1sIHZlcnNpb249IjEuMCIgZW5jb2Rpbmc9IlVURi04IiBzdGFuZGFsb25lPSJ5ZXMiPz4gC
jxGYWtlU2VjdXJpdHkgeG1sbnM9Imh0dHA6Ly93d3cub3Blbmdpcy5uZXQvQ29kZVNwcmludDIwMjNP
Y3QvU2VjdXJpdHkiPiAKICAgIDxGYWtlTGV2ZWw+VE9UQUxMWSBTRUNSRVQ8L0Zha2VMZXZlbD4KICA
```

```
gIDxGYWtlQ2F2ZWF0PkNyb3AgaXQgbGlrZSBpdCdzIGhvdC48L0Zha2VDYXZlYXQOy4O0tEBmtq9YVV
tqiXOLq6pfO/liXQNZ9muB9K5h8K2Uxkb48DBst6LdWwUHu1ZaUYupq2LqmVMyqJjUB5JNXhfG73nEW
XAhwEtoRSQ8bS9SWgxfIjIELput9gWU0K6/
            (..TRUNCATED for BREVITY..)
            xBzMXU3ecl0VCQk02dYsggLfbPY0kBSH3wDMORoHSfPUcQf4s4pBnilqQacZoPiH
vC+S0wdblK9+C3+Uyvb8ihKYcOg44t9wSlD+G6iB6CeceKS7TmpzOZ0FkRv5qmvU0n5vGLbPw8ci
+HjWbgkjmvbtMy+YdplAF7PwHmx9M7uAq7BO86NrMk451Nl786M54VDLwXko6kXNy1ryaze78Wdi
JxtvQDR7bSOLHUaEoxY1Pq40pRBmG390knItLsb2yBga4UjNiHJqFTykPBXpwE0KMRhOdbW5rEuI
2PewliPDDACc+mAEd3bh5kI7Et2rM6kSgOorVpnREeNSF5sB1ofr6nfluzBh1jXAvPA92lFv7USm
DH0YDUy7oyuBsH4gJkcQCu0wuUPo2EWJxYx65sf7gpEt3OW4jMzcl5500w3RSI2kEJTdOfiv+wLBl
WM2zTFLaWK4eJrq9Lh/IqOWwO9DW7wnG987Bq7xnlDHXLHZfA5z6wU11iDAZxBiWwhYbFGk0H+a/
uNj748wcg2FZ3ZUwDVSB+/fRl7H4eAlxciDoUMw6KxPWatpnEdTEiWXvSFYiDoPwQ4rOU2jv7BKyxpL
LozcYQRYGzcA0YEpBMqZEgiaQUqTKqi+tyWa1wKD13ZCHvapqc6A+0G4pMyewS+pAC35AsiAnawUcSO
8TeKvk2M/oi6MAbgVzXrUrRLM+4+K3kv3cM4rES9q/C+M4Iu42XvF3+hMMLecOP7xMW82XggIC3aNCb
05ks+XYih+rL7KEHnJZuxR2dmUSWZprJbkAZEXUPYWXymyXJVhk6Bc3m5llRX9SND5hailb9o5aFem2
mHfKs5lqt7/DApcZBaP05VAiVLLU+GX/cYJnQ0rDHsE4S0aGosDXE0RkwKIXs86WYYoYHPeOzH4xTvG
8TreBAyxAFlvpErWwSwfmjKA"
        }
}
```

If successful, the response will be as shown below.

```
{
    "id": "result",
    "value": "Storage successful"
}
```

Figure 39 shows an image sent using postman (right hand-side), and stored (lower-left), through a process in pygeoapi (upper-left).

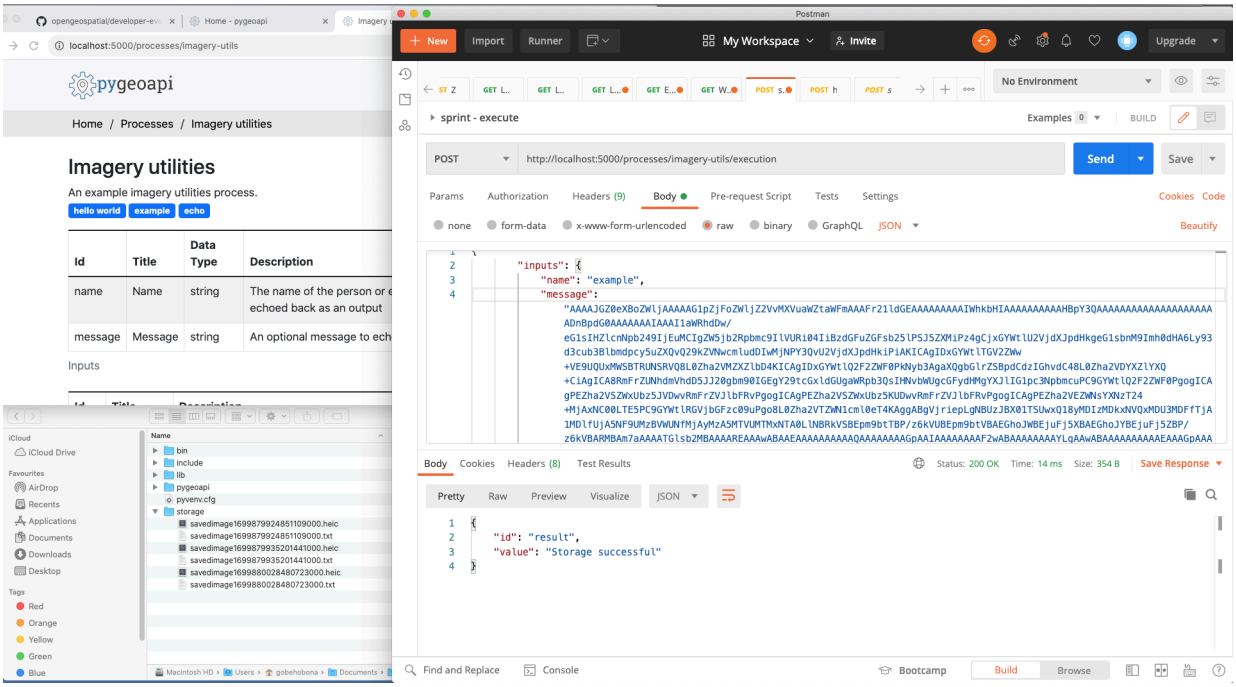


Figure 39 — Screenshot showing an image sent using postman (right hand-side), and stored (lower-left), through a process in pygeoapi (upper-left)

One of the observations from the prototype is that the approach of sending base64 encoded bytes via a POST request is appropriate for small images. However, for large images, such as

motion imagery collected over several hours, it may be necessary to send a request that includes a URL from which the server may stream the data.

During the three-day code sprint, participants were able to implement tools for creating GIMI files. Multiple sample files were created as a result. However, due to time constraints, it was not possible to integrate the tools used for creating the GIMI files into a Web API.

4.4.3. Towards open source libraries for GIMI

This section presents findings of prototyping GIMI support in a selection of open source libraries. The prototyping and assessment was carried out by Brad Hards.

Prior to the code sprint, a selection of GIMI files were produced using a custom application. The application used open data provided by the Australian Capital Territory (ACT) Government under a CC-BY license. The source format was GeoTIFF, and the resulting GIMI files used HEVC (H.265), JPEG 2000, and uncompressed encodings. Non-standard JPEG and AV1 (AVIF) files were also produced, along with a sample that did not include security metadata and a sample that did not include corner points.

Additional files were produced during the sprint from Synthetic Aperture Radar (SAR) data in NITF-formatted Sensor Independent Derived Data (SIDD) format, sourced from Capella Space via the AWS Open Data Program, also under a CC-BY license.

These GIMI files were read back in using modified versions of libheif and GDAL, where GDAL already used libheif for image content, but required an external “sidecar” PAM file to provide any geopositioning or associated metadata.

The changes to libheif were to support metadata that is not attached to an image item. This occurs in GIMI files for the security metadata, which is scoped to the entire HEIF file content. The changes included suppressing a parsing error and exposing the resulting metadata item through the libheif API.

The changes to GDAL were to support the security metadata as a GDAL metadata item, and to use the KLV metadata as a GDAL metadata item, and also to provide corner point geopositioning. The changes to libheif enabled access to the security metadata. With these changes, GDAL was able to extract and expose the additional GIMI metadata without use of sidecar files. An example showing the GIMI-specific metadata domains, along with GCPs and the derived coordinate system transformation, is shown below.

```
$ gdalinfo -mdd all ~/ogc-developer-events/2023/Open-Standards-Code-Sprint/
publicly-releasable-sample-data/bradh/ACT2020_wgs_84_trimmed.heif
Driver: HEIF/ISO/IEC 23008-12:2017 High Efficiency Image File Format
Files: /home/bradh/ogc-developer-events/2023/Open-Standards-Code-Sprint/
publicly-releasable-sample-data/bradh/ACT2020_wgs_84_trimmed.heif
Size is 2048, 1024
Coordinate System is:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
```

```
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
Origin = (149.102048999999994,-35.315807000000000)
Pixel Size = (0.000001009277344,-0.000001009765625)
GCP Projection =
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
GCP[  0]: Id=GCP_1, Info=
          (0,0) -> (149.102049,-35.315807,0)
GCP[  1]: Id=GCP_2, Info=
          (2048,0) -> (149.104116,-35.315807,0)
GCP[  2]: Id=GCP_3, Info=
          (2048,1024) -> (149.104116,-35.316841,0)
GCP[  3]: Id=GCP_4, Info=
          (0,1024) -> (149.102049,-35.316841,0)
Metadata (GIMI File Security):
  XML=<?xml version="1.0" encoding="UTF-8" standalone="yes"?><FakeSecurity
xmlns="http://www.opengis.net/CodeSprint2023Oct/Security"><FakeLevel>SECRETIVE-
ISH</FakeLevel>
<FakeCaveat>ButterPopcorn</FakeCaveat><FakeCaveat>LowPlaces</FakeCaveat>
<FakeRelTo>US</FakeRelTo><FakeRelTo>AUS</FakeRelTo><FakeRelTo>UK</FakeRelTo>
<FakeDeclassOn>2024-11-01</FakeDeclassOn></FakeSecurity>
Metadata (GIMI ST0601):
  Checksum=0x5f 0x77
  Corner Latitude Point 1 (Full)=-35.315807
  Corner Latitude Point 2 (Full)=-35.315807
  Corner Latitude Point 3 (Full)=-35.316841
  Corner Latitude Point 4 (Full)=-35.316841
  Corner Longitude Point 1 (Full)=149.102049
  Corner Longitude Point 2 (Full)=149.104116
  Corner Longitude Point 3 (Full)=149.104116
  Corner Longitude Point 4 (Full)=149.102049
  Mission ID=Mission3
  Precision Time Stamp=1589944740032704
  ST 0601 Version=ST 0601.19
Metadata (DERIVED_SUBDATASETS):
  DERIVED_SUBDATASET_1_NAME=DERIVED_SUBDATASET:LOGAMPLITUDE:/home/bradh/ogc-
developer-events/2023/Open-Standards-Code-Sprint/publicly-releasable-sample-
data/bradh/ACT2020_wgs_84_trimmed.heif
  DERIVED_SUBDATASET_1_DESC=log10 of amplitude of input bands from /home/
bradh/ogc-developer-events/2023/Open-Standards-Code-Sprint/publicly-releasable-
sample-data/bradh/ACT2020_wgs_84_trimmed.heif
Corner Coordinates:
Upper Left  ( 149.1020490, -35.3158070) (149d 6' 7.38"E, 35d18'56.91"S)
```

```
Lower Left  ( 149.1020490, -35.3168410) (149d 6' 7.38"E, 35d19' 0.63"S)
Upper Right ( 149.1041160, -35.3158070) (149d 6'14.82"E, 35d18'56.91"S)
Lower Right ( 149.1041160, -35.3168410) (149d 6'14.82"E, 35d19' 0.63"S)
Center      ( 149.1030825, -35.3163240) (149d 6'11.10"E, 35d18'58.77"S)
Band 1 Block=2048x1 Type=Byte, ColorInterp=Red
Band 2 Block=2048x1 Type=Byte, ColorInterp=Green
Band 3 Block=2048x1 Type=Byte, ColorInterp=Blue
```

GDAL serves as a key component within the geospatial software ecosystem, with broad direct and indirect use by a wide range of open source and proprietary software. A view of the ecosystem is shown below, with components higher in the stack reliant on capabilities provided by components lower in the stack.

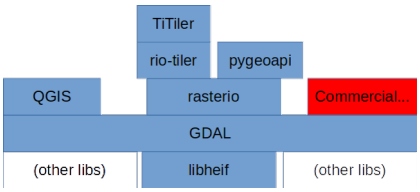


Figure 40 — Illustration of the ecosystem

As noted above, libheif and GDAL were modified to expose GIMI features within existing GDAL interfaces. This allows use of these features by existing software, often without change. An example is shown in Figure 41, which is a screenshot of an unmodified version of QGIS displaying two GIMI test files (using different compression methods), overlaid on top of OpenStreetMap. A small script is running in the QGIS Python console to extract the XML-encoded security metadata and provide an on-screen, human-readable, color-coded representation of the key aspects of the security data.

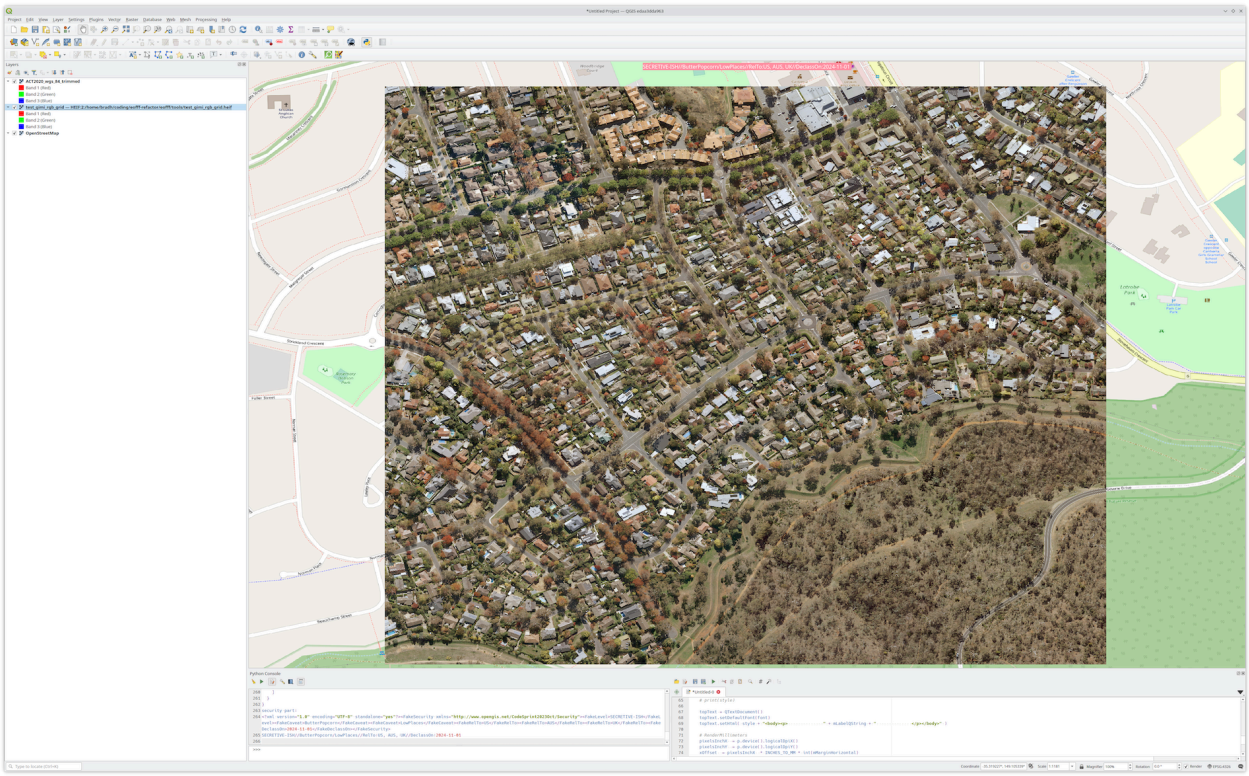


Figure 41 — Screenshot of a test GIMI image displayed in QGIS

Similarly, rebuilding rasterio against the modified GDAL library makes the GIMI features available within a Python context. The sprint participants created a Jupyter Notebook, shown in Figure 42 and Figure 43, that used rasterio to perform basic analysis / data science on GIMI imagery files. Note that this required development of the notebook, but there were no changes to rasterio or to the general design of the Juypter notebook specific to GIMI or HEIF. Instead, the abstraction provided by GDAL meant that the notebook could basically follow the rasterio introductory tutorial.

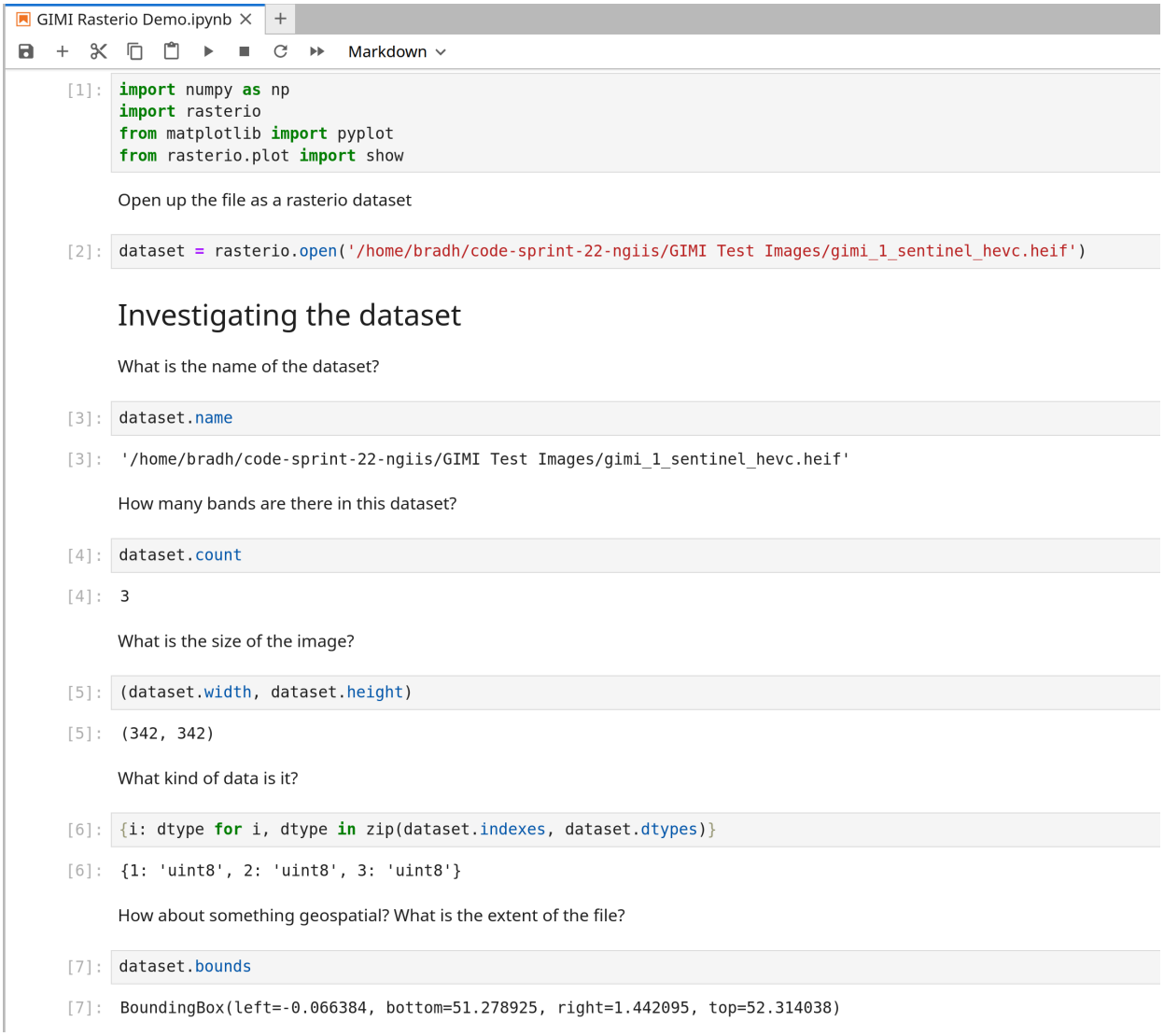


Figure 42 — Screenshot of a jupyter notebook for displaying a GIMI file

```
[10]: dataset.colorinterp
```

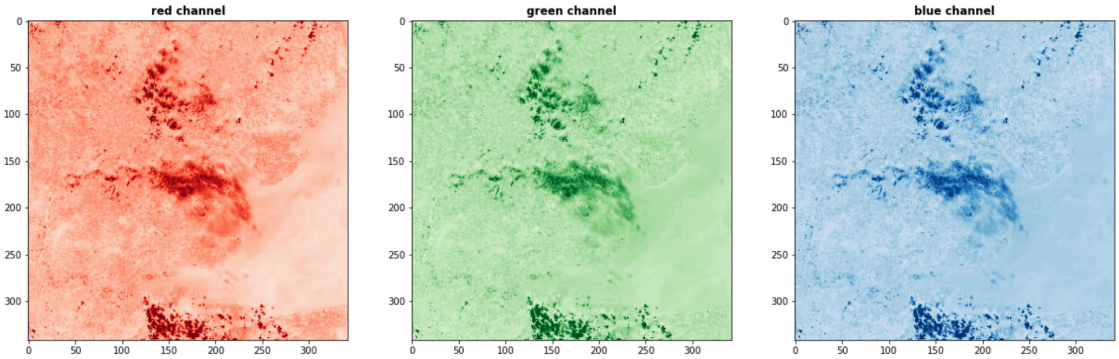
```
[10]: (<ColorInterp.red: 3>, <ColorInterp.green: 4>, <ColorInterp.blue: 5>)
```

```
[11]: red = dataset.read(1)
```

```
[12]: green = dataset.read(2)
```

```
[13]: blue = dataset.read(3)
```

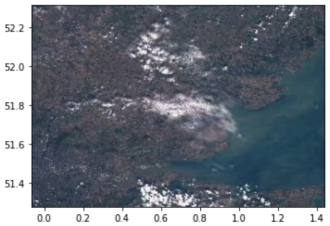
```
[14]: fig, (axr, axg, axb) = pyplot.subplots(1,3, figsize=(21,7))

show(red, ax=axr, cmap='Reds', title='red channel');
show(green, ax=axg, cmap='Greens', title='green channel');
show(blue, ax=axb, cmap='Blues', title='blue channel');
```



All together now...

View the data as RGB with geospatial labels

```
[15]: show(dataset);
```



```
[16]: red[0,0]
```

```
[16]: 86
```

Figure 43 — Second screenshot of a jupyter notebook for displaying a GIMI file

In addition to custom development, there are a wide range of open source geospatial tools written in Python that make heavy use of rasterio for raster processing. An example of this is pygeoapi. Installing pygeoapi with the local copy of rasterio was required to ensure that a second copy of the rasterio stack (including GDAL) was not used. Configuring pygeoapi with a GIMI file as a dataset was the only other activity required to serve the GIMI file in a range of formats, including as an OGC API — Coverages endpoint. A screenshot of the bounds, domain set and ranges of a coverage from a GIMI file on pygeoapi is shown in Figure 44.

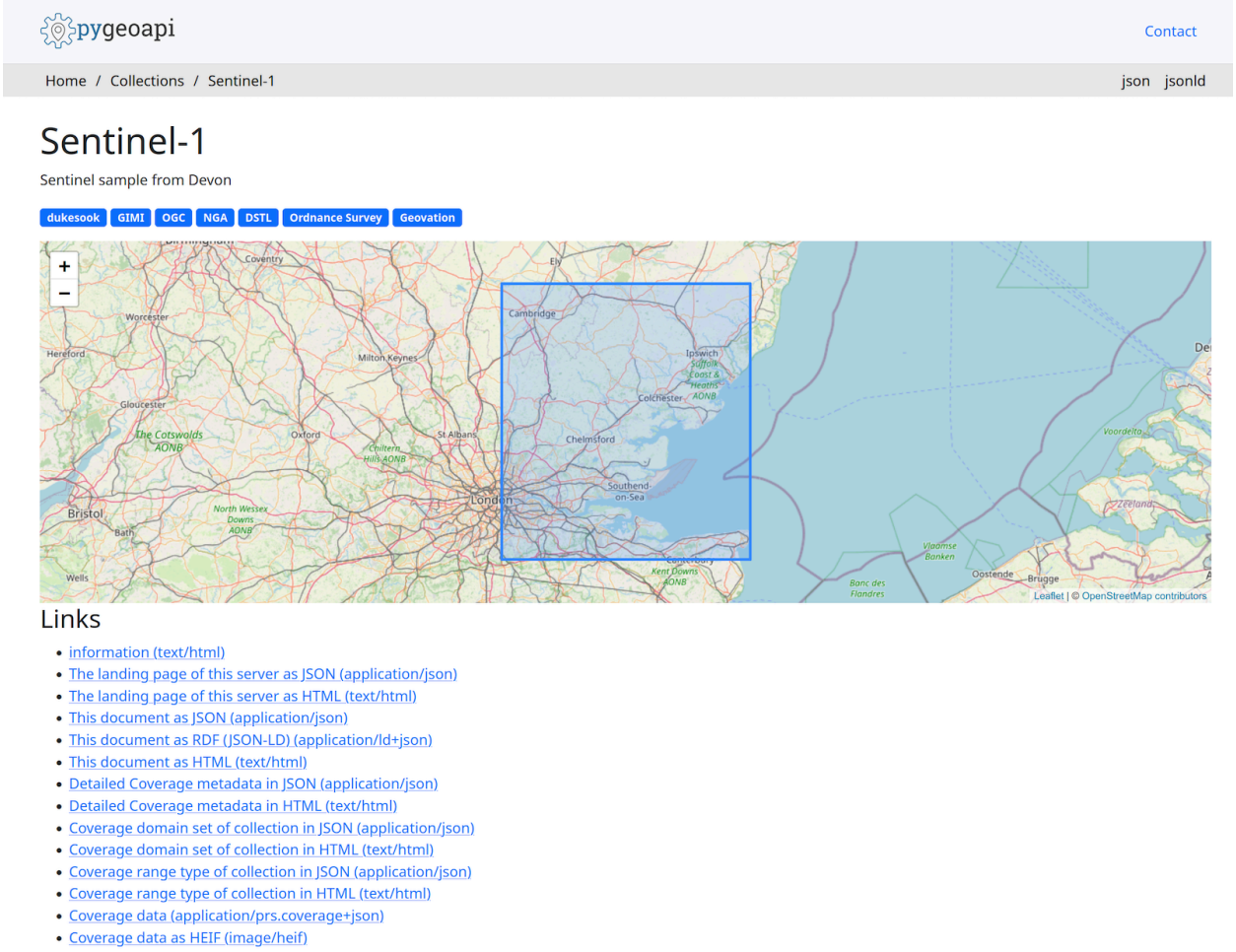


Figure 44 — Overview of bounds, domain set and ranges of a coverage from a GIMI file on pygeoapi

That coverage endpoint could then be used with the open-source TiTiler application to dynamically generate tiles at various zoom levels. Like pygeoapi, TiTiler also uses rasterio, and only required minor install-time customization to use the local copy of the rasterio stack (including the modified GDAL and libheif versions). No code changes were required. With this installation, TiTiler served raster tiles in a range of formats, including as an OGC API — Tiles endpoint.

Those raster tiles could then be displayed in a web map such as OpenLayers as shown in Figure 45.

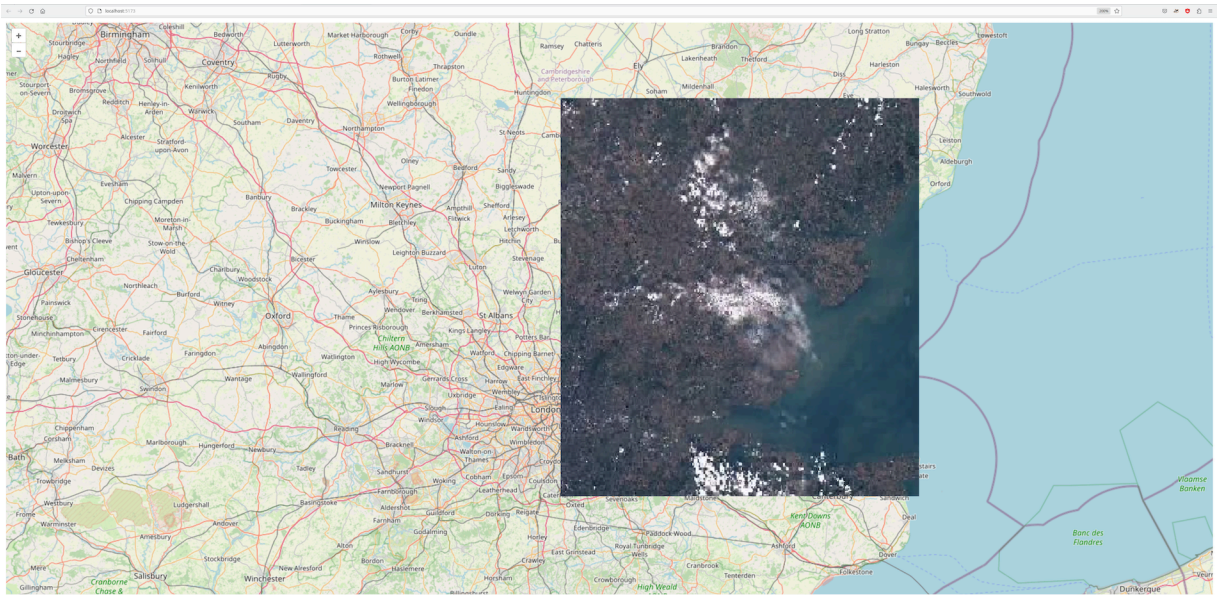


Figure 45 — GIMI image file displayed in OpenLayers

In conclusion, the sprint participants found that with some changes to libheif and GDAL, a wide range of software can quickly and simply make use of GIMI capabilities, including making those available in standards-compliant OGC API services. However, to fully unlock those GIMI capabilities, the base libraries will need to ship with the required functionality “out of the box”, including:

- comprehensive codec support (noting that AVC/H.264 and High Throughput JPEG 2000 are not supported in libheif at this time);
- a finalized version of the geolocation metadata;
- flexible support for security metadata;
- support for overviews, grid representation, and byte range requests (equivalent to Cloud Optimized GeoTIFF);
- GIMI writing support within GDAL;
- unit testing and integration testing with a wider range of sample data; and
- a GIMI validator.

4.4.4. Testing Windows open source support for GIMI

This section presents findings of prototyping GIMI support within Windows using Python and open source libraries. The prototyping and assessment were carried out by Samantha Lavender.

The first part of the experiment was to try and read the metadata from within GIMI files using Python. The GitHub repository was searched and the pyisobmff repository was found. This was forked and a branch created so that testing and updates could be undertaken.

First, the Python code was updated so it ran for the example GIMI files requiring a change in one line, from extent to item:

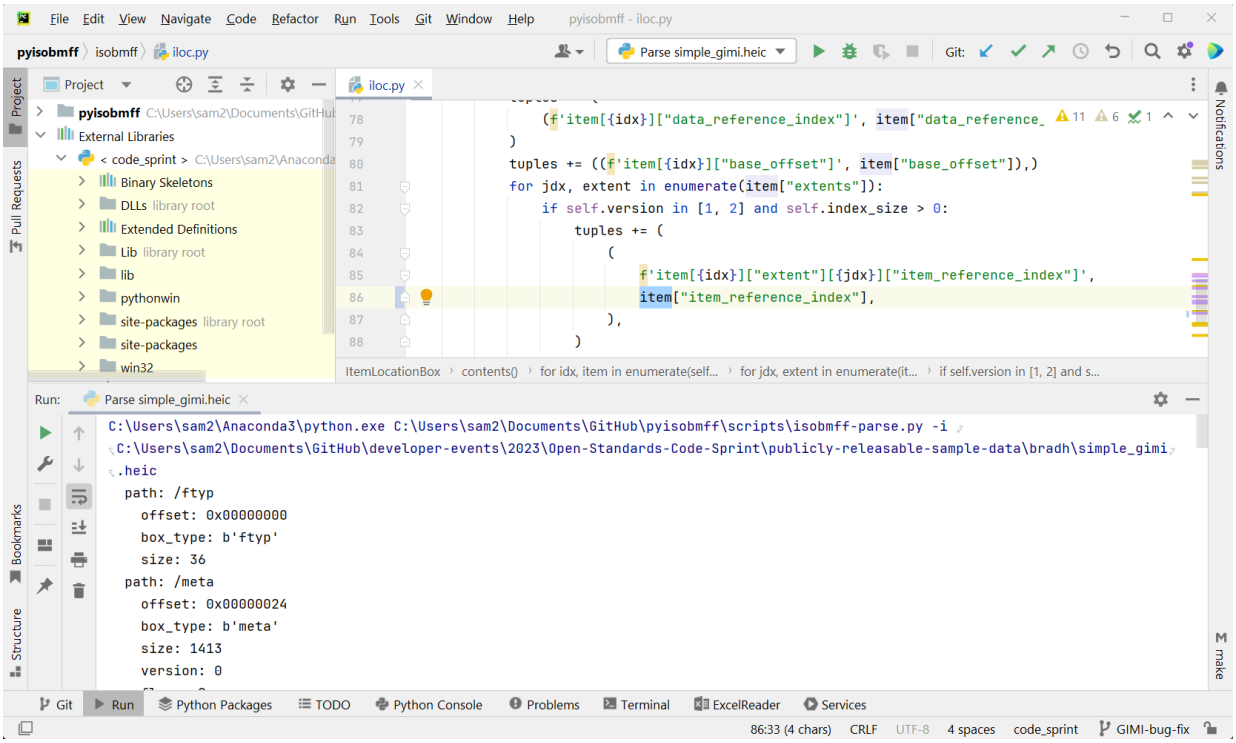


Figure 46 — Code change in Python script for parsing GIMI file with pyisobmff

Then, the next experiment focused on reading the images. For this, the libvips image processing library was tested. On Windows, pre-compiled DLLs are needed for the Python library pyvips, and they should be specified in the Windows Advanced System Settings control panel, or via the PATH in the Python code. The second approach was used:

```
import os
vipsbin = r'C:\DDrive\OGC-code-sprint\vips-dev-8.15\bin'
os.environ['PATH'] = vipsbin + ';' + os.environ['PATH']
```

However, by default, the compiled DLLs do not come with the support for the needed codec compression. Therefore, it was decided to recompile the DLLs following the option described in the build-win64-mxe repository.

Through trial and error, it was found that the easiest way to compile the DLLs was to:

1. install Windows Docker Hub and link it to Ubuntu installed on a Windows Subsystem for Linux (WSL); and

2. then, the bash script could be run, which downloads a Docker image, builds a libvips Docker image, and then compiles all the DLLs required for pyvips:

   ```
   ./build.sh all --with-hevc
   ```

This allowed the HEIF and HEIC examples to be read, but not the HEJ2 sample. The next experiment undertaken was to update build-win64-mxe to support HEJ2 as this is supported in the latest version of libheif, but in the time available it was not achieved.

The code in the pixalytics pyisobmff branch was updated to add an option using the function extract_items to read the images and output them as JPEG files:

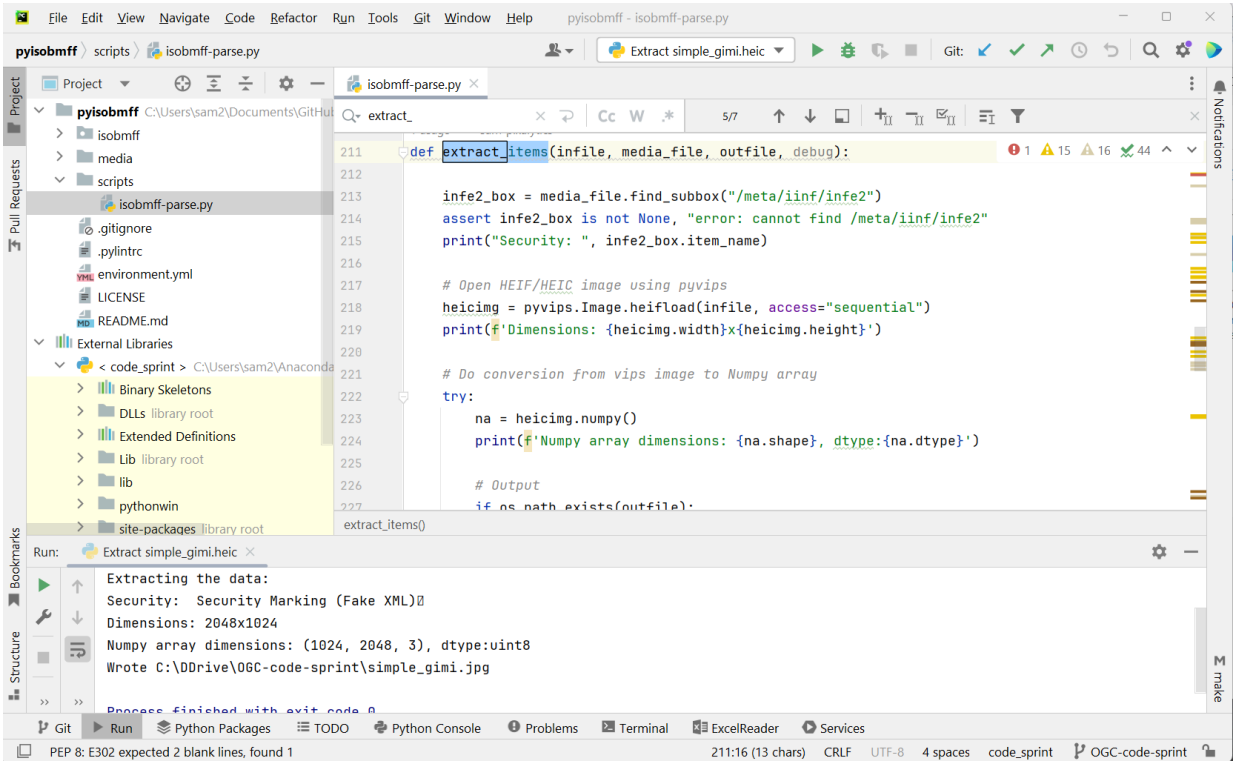


Figure 47 — Using pyisobmff for reading GIMI image

4.4.5. GIMI web-map client

Some of the processing needed to display GIMI imagery in a web browser can be performed on either the server or the client. In the former case, a (web) server has to parse the metadata and decode the image, sending separate pieces of information to the web browser.

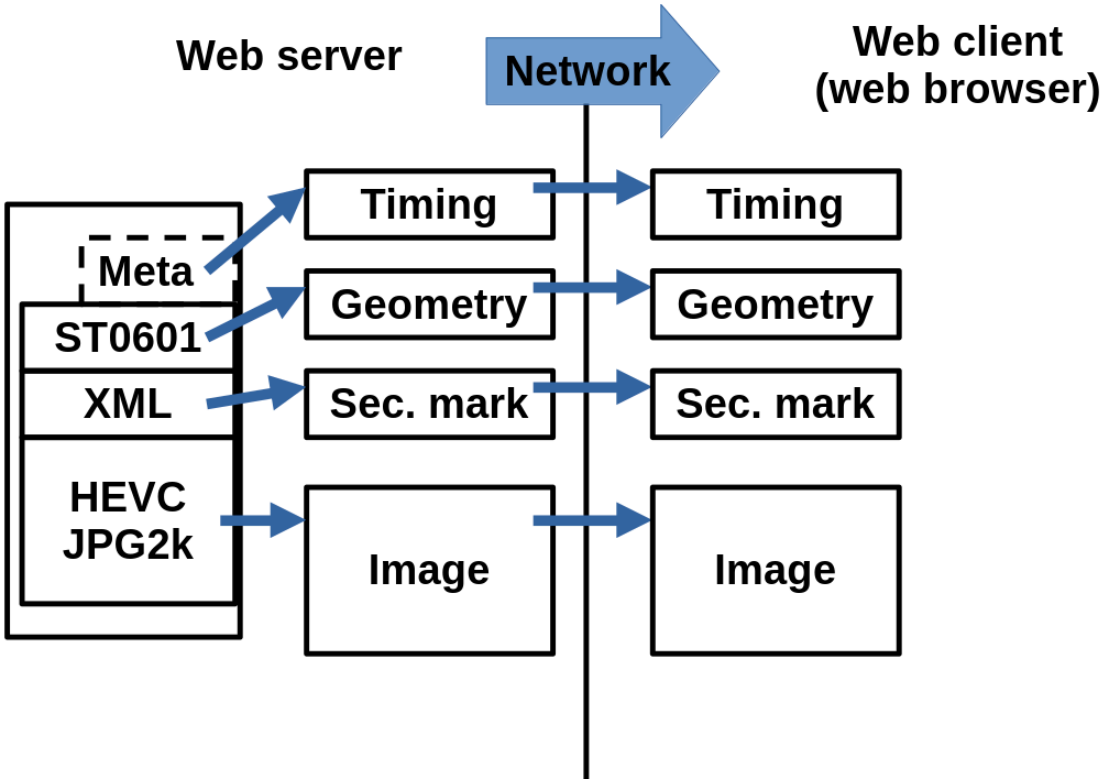


Figure 48 — Schematic data flow of server-side GIMI processing for display

In the latter case, the web browser receives the entire ISOBMFF container file, parses the metadata, and extracts and decodes the image.

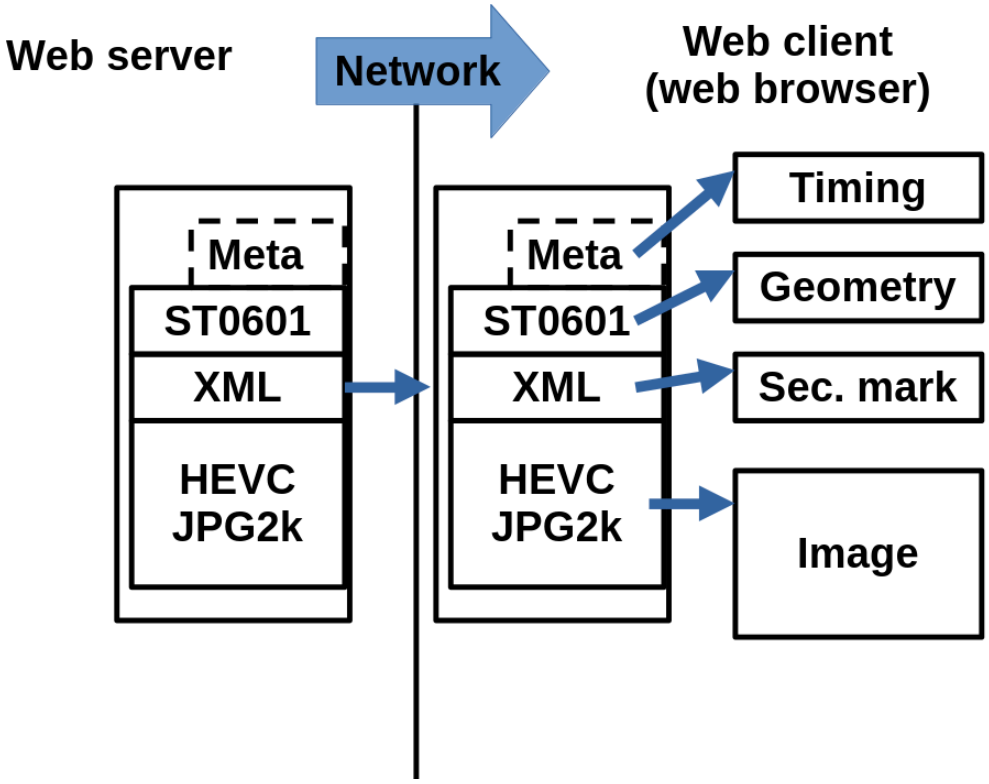


Figure 49 — Schematic data flow of client-side GIMI processing for display

There is a need to compare both approaches (in terms of flexibility, performance, integrity, etc.), as well as considering hybrid approaches; but no client-only GIMI implementation existed prior to the code sprint. Therefore, sprint participants prototyped a client application, called web-gimi-viewer, for displaying GIMI imagery. Development of web-gimi-viewer helped to identify a number of issues relating to GIMI.

One of those issues was that there are records of metadata in various parts of a GIMI file, and each record of metadata uses a different encoding. This is illustrated in Figure 50. The four kinds of (meta)data in different encodings therefore require four different tools to be parsed, and thus four different blocks of code (which duplicates effort, cost, and possible software defects).

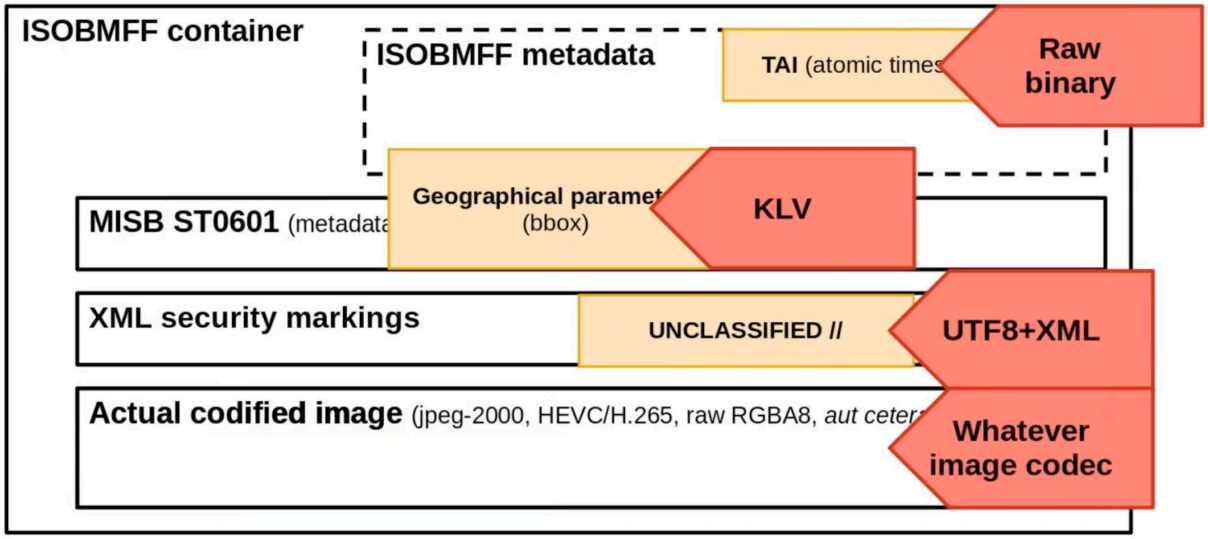


Figure 50 — Metadata encodings found in a GIMI file

The ISOBMFF structure of the file is handled with the mp4box library. A few technical issues within this library were discovered and fixed during the code sprint.

Geographical information is currently encoded in KLV conforming to MISB ST 0601, which is a standard for transmitting the attitude and status of an unmanned aerial vehicle via radio link, in real time, to a ground station. This encoding is handled with vidterra's misb.js library, which needed substantial adaptations to work in a web browser.

Instead of KLV+ST0601, it might be preferable to store geographical information directly in the ISOBMFF metadata fields using a WKB (Well-Known Binary) encoding. The non-domain-specific GeoPose OGC Standard should also be considered. Further research would be needed to explore the technical viability of WKB and/or GeoPose when a GIMI file contains video instead of a single still image.

Security markings are encoded in a small XML document. While parsing XML in a web browser is trivialized due to built-in functionality, this is not the case in server implementations. Therefore, it will be beneficial to consider ways of implementing Dublin Core and/or ISO 19115-1:2014 metadata alongside, possibly encoded in JSON.

Atomic time metadata is stored in ISOBMFF metadata fields and is decoded with no need of an external library since its structure is relatively simple.

The simplest version of web-gimi-viewer offers a text dump of all the aforementioned metadata, as shown in Figure 51.

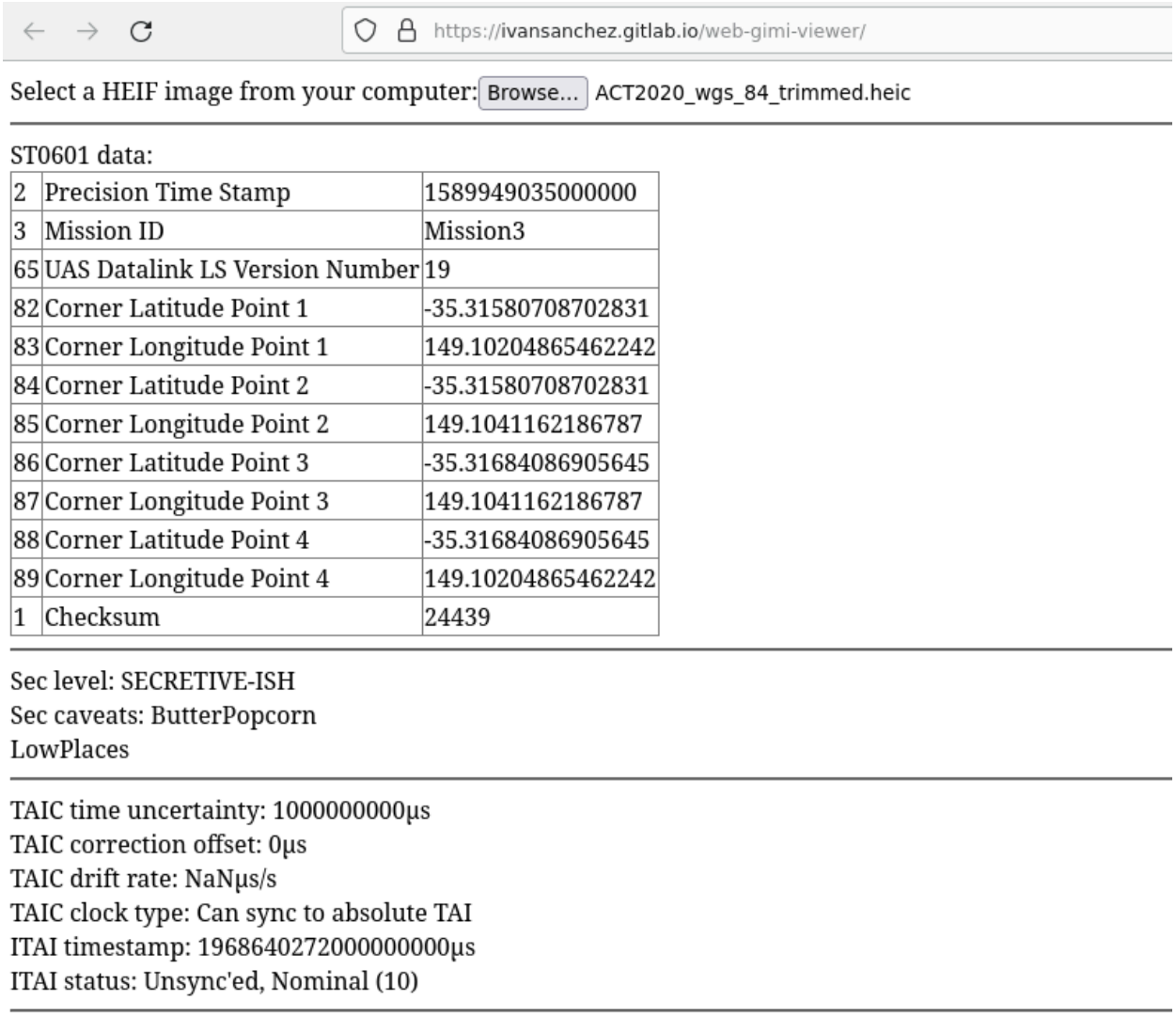


Figure 51 — Screenshot of the prototype web-gimi-viewer displaying metadata from a GIMI file in text form

A slightly more advanced version of web-gimi-viewer displays the same metadata on a map interface, as shown in Figure 52.

The sprint participants noted that from a web browser perspective the tools and libraries needed included an ISOBMFF parser (e.g., mp4box.js), KLV+ST0601 parser (e.g., misb.js), XML decoder (built-in), TAI decoder, and image codecs (e.g., heif2any). The prototype web-gimi-viewer was implemented using the aforementioned libraries. A screenshot of the viewer is shown in Figure 53.

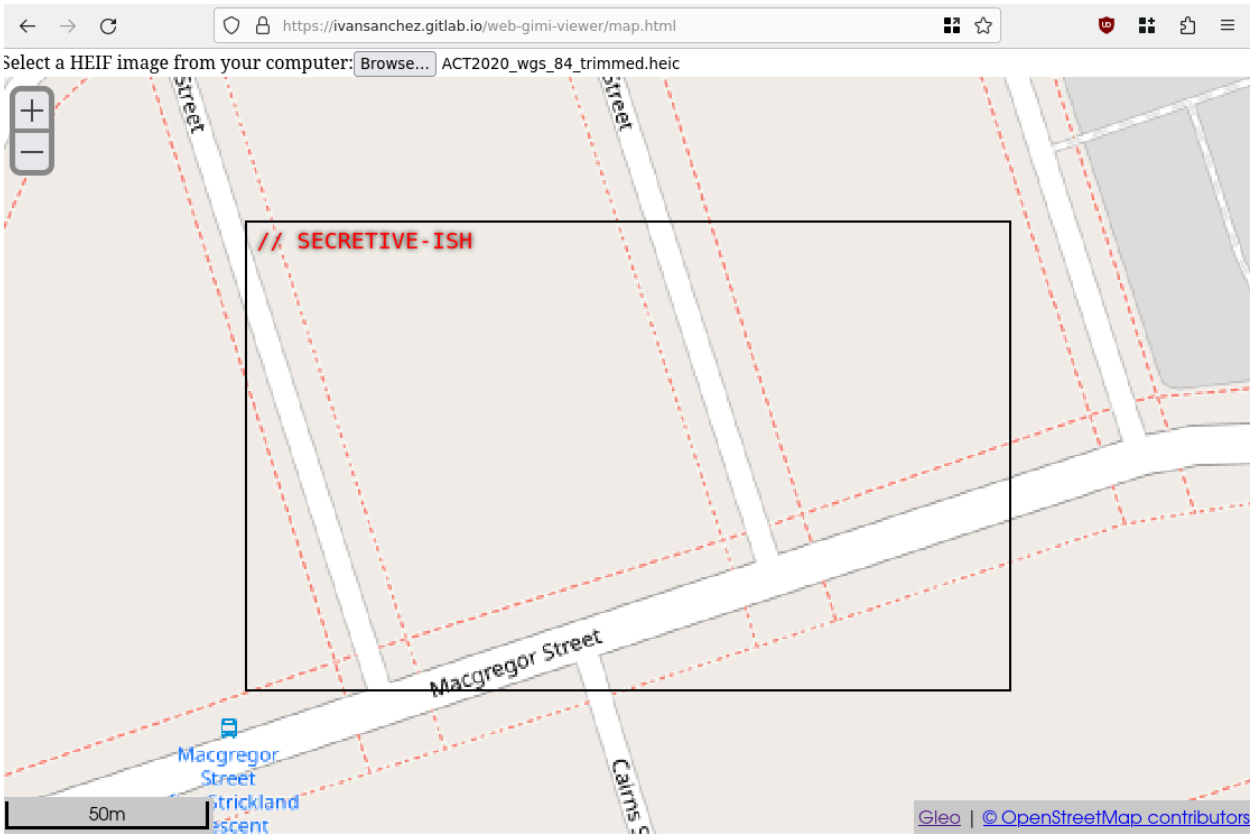


Figure 52 — Screenshot of the prototype web-gimi-viewer displaying security and TAI metadata from a GIMI file

Web browsers have native support for a very limited number of image formats/codecs, and supporting additional codecs requires explicit effort in the web application. Support for the HVC1 codec could not be implemented due to time constraints, this being the reason why Figure 52 displays an empty rectangle instead of the imagery.

By contrast, the file in Figure 53 uses the HEIF codec, which can be decoded with the aid of the heic2any library.

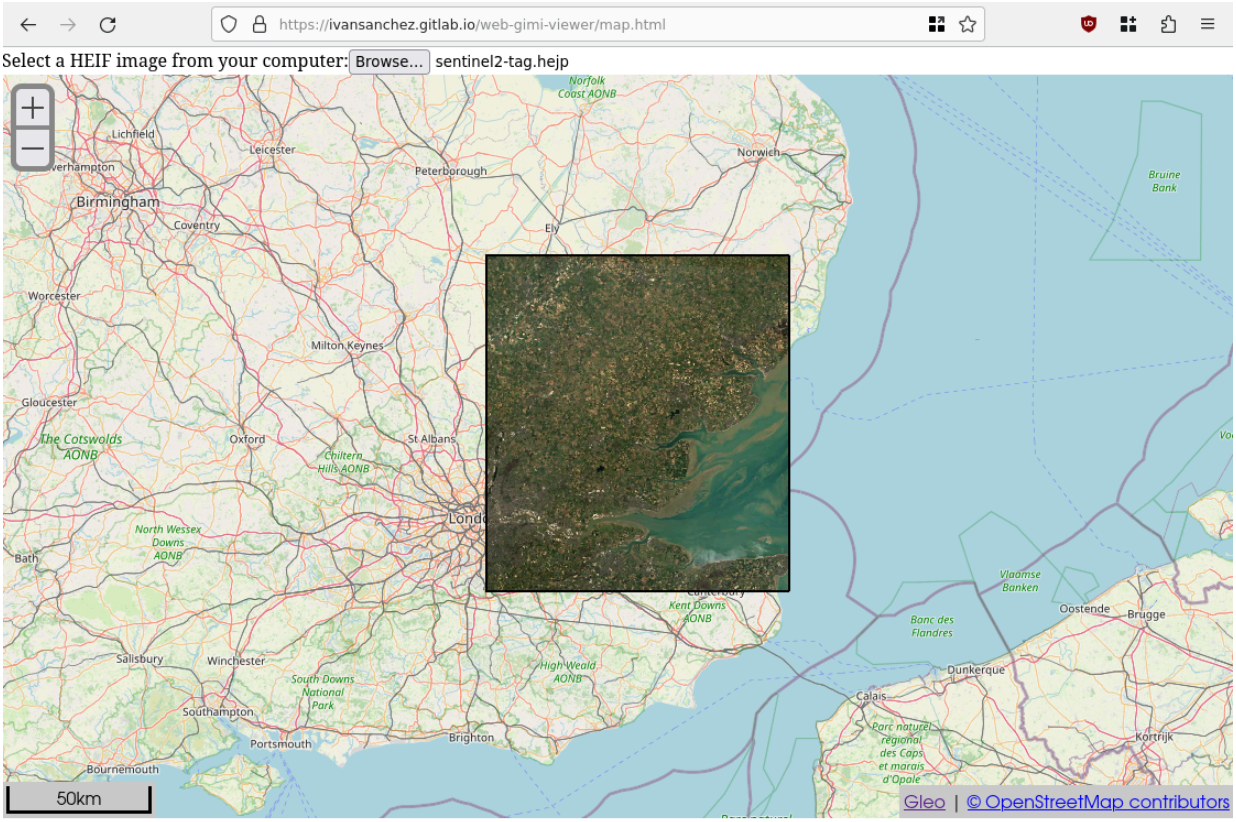


Figure 53 — Screenshot of the prototype web-gimi-viewer displaying a GIMI file with HEIF imagery. Contains modified Copernicus Sentinel data 2023.

4.4.6. Ecere GNOSIS Map Server

The Sprint participants from Ecere collaborated with those from UAB-CREAF to compare results of the GNOSIS Map Server vs. the CREAF MiraMon server and also drafted examples, using GNOSIS and MiraMon as example implementations.

During the Sprint, support for the HEIF output format in GNOSIS Map Server was integrated using the libheif library, including support for uncompressed RGB(A) images, as well as JPEG 2000 and JPEG encoding inside the HEIF file.

This enabled GNOSIS Map Server to offer GIMI-conformant images through interfaces that conform to the OGC API — Maps candidate Standard and tiled images through an interface conforming to the OGC API — Tiles Standard.

Work on implementing support for GIMI through an interface conforming to OGC API — Coverages on GNOSIS Map Server was still in progress by the end of the code sprint due to additional work required to support raw floating point or integer data values and varying number of bands.

Other work accomplished on support for GIMI on GNOSIS Map Server included enabling HEIF content negotiation through the `f` query parameter or `Accept` header used in implementations

of OGC API Standards. Content negotiation is a process that enables a client and server to determine the encoding that the server should return a response in. A table of the supported encodings is provided in Table 1.

Table 1 — Mapping of HEIF media types to OGC API content negotiation

ENCODING	ACCEPT:	?F=
JPEG 2000	image/hej2k	hej2
JPEG	image/hejpg	hejp
Uncompressed	image/heif	heif

Implementation of support for GIMI in GNOSIS made it possible to identify a number of bugs / limitations in libheif that were quickly fixed by Dirk Farin (a contributor to libheif). The bugs/ limitation were:

- invalid output encoding as JPEG with alpha channel;
- color conversion issues encoding as JPEG 2000;
- lack of support for encoding as JPEG 2000 with alpha channel;
- lack of support to set URI type on URI metadata items; and
- lack of support for various low-level metadata modification capabilities.

The team from Ecere carried out an initial investigation and had some success in encoding ST0601 KLV metadata in HEIF file (using example hardcoded KLV metadata for imagery in the UK). It was determined that additional work would be required to encode complete geospatial referencing in the KLV metadata based on the request.

The team collaborated with other GIMI participants to provide an OGC API — Maps endpoint for the CREAF MiraMon client demonstration, and sample images from Sentinel-2 data produced by the GNOSIS Map Server for the web-gimi-viewer demonstration (including the KLV metadata).

Some of the input images used within this code sprint to create GIMI test files were Sentinel-2 imagery, with GMLJP2 information embedded. One of the observations made by sprint participants from Ecere was that the conceptual model used by GMLJP2 Standard could be specified as a separate standard to make it possible to use on other ISOBMFF profiles, including GIMI. Currently GMLJP2 has been designed specifically for use with JPEG 2000 imagery. One of the most well-known implementations of GMLJP2 is the Copernicus Sentinel-2 mission, which is funded by the European Union and operated by the European Space Agency (ESA).

4.4.7. GIMI support in the MiraMon Web Browser

Sprint participants from UAB-CREAF worked on the implementation of a web client that can access NGIIS data through files in a GIMI profile or through an OGC API — Maps interface to show the metadata and display the imagery in a web browser.

4.4.7.1. Accessing GIMI data through a UAB-CREAF web client

The sprint participants began developing a simple client that is capable through an HEIF file or URL to read the boxes to get the data and metadata to show the timing, content id, bounding box, and security information and to display the imagery in a web browser. The client is capable of reading HEIF files with JPEG, JPEG 2000, and HEIC images format. Note that HEIC refers to HEIF images containing HEVC-encoded images (i.e., H.265 encoded images).

The client uses several JavaScript libraries:

- “mp4box” library to read the HEIF file, construct the boxes, and get the items and the data;
- “vidterra’s” library adapted to a ST0601 Metadata format by Iván Sanchéz during the code sprint to provide the metadata information;
- “openjpeg” library to convert the JPEG 2000 images into an object that can be used to construct an “ArrayBuffer” to display the image into a canvas; and
- “heic2any” library to convert the HEIC images to a JPEG/PNG format.

The client uses a native HTML method to display JPEG/PNG images, codifying the image in base64 in an IMG element.

File Navega... gimi_1_sentinel_hevc.heif Load

FILE INFORMATION

INFORMATION FOLLOWING THE ITEMS

FileTypeBox(ftyp)

Major Brand: heic
Compatible Brands: mif1, heic, geo1, unif, miaf
Handler type: pict

Fake Security XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FakeSecurity xmlns="http://www.opengis.net/CodeSprint2023Oct/Security">
    <FakeLevel>UNRESTRICTED</FakeLevel>
    <FakeCaveat>Do not feed the programmers. They bite.</FakeCaveat>
    <FakeCaveat>File may contain traces of brilliance and sarcasm.</FakeCaveat>
    <FakeRelTo>JP</FakeRelTo>
    <FakeRelTo>AU</FakeRelTo>
    <FakeDeclassOn>2023-10-29</FakeDeclassOn>
</FakeSecurity>
```

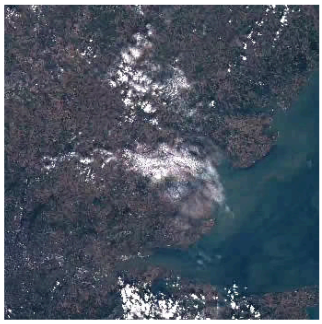
Content ID for Fake XML: urn:uuid:75E544E6-6E15-4A3A-BEF2-C3C9C9AFD9CD

Image
type: hvc1
width: 342
height: 342
Number of channels: 3
Bits per channel: 8,8,8
Content ID for Image: urn:uuid:87015BD8-C84B-4778-B593-5EF16982E452

ST0601 Metadata
Precision Time Stamp: 1694775541999918 µs
Mission ID: S2A_MSIL1C_20230915T105701_N0509_R094_T30UYC_20230915T131504.SAFE
Corner Latitude Point 1: 52.31403779346218°
Corner Longitude Point 1: -0.06638425405435555°
Corner Latitude Point 2: 52.31403779346218°
Corner Longitude Point 2: 1.442095209584636°
Corner Latitude Point 3: 51.27892518009941°
Corner Longitude Point 3: 1.442095209584636°
Corner Latitude Point 4 : 51.27892518009941°
Corner Longitude Point 4: -0.06638425405435555°

Bounding Box: {"MinX":-0.06638425405435555,"MaxX":1.442095209584636,"MinY":51.27892518009941,"MaxY":52.31403779346218}
IMAGE

UNRESTRICTED



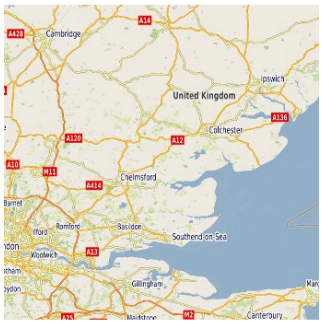


Figure 54 — Screenshot of the web interface showing the data and image extracted from the heif file

4.4.7.2. Accessing GIMI data through a MiraMon Web Map Browser

The UAB-CREAF Sprint participants implemented an extension to the MiraMon Web Map Browser that allows access to data in GIMI (HEIF) format from an OGC API — Maps server, as well as the ability to display and operate on the obtained images.

The map browser connects to an OGC API — Maps interface with HEIF images and makes the necessary requests to get the images, display, zoom, pan them, etc. The image in HEIF format following the GIMI profile obtained as a response to each request is converted into a binary

array allowing the display of the data on a canvas, selection of the symbolization and operation on the data by making calculations, histograms, etc.

During the code sprint, the sprint participants from CREAF used the OGC API — Maps server implemented by Jerome Jacovella-St-Louis (Ecere) and deployed at https://maps.gnosis.earth/ogcapi.

This is a sample of a map request in HEIF format to the gnosis server: https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?crs=EPSG:4326&bbox-crs=EPSG:4326&bbox=46.645009215,-18.52545820,59.41112249,14.65325273&width=1510&height=581&f=hej2&transparent=false

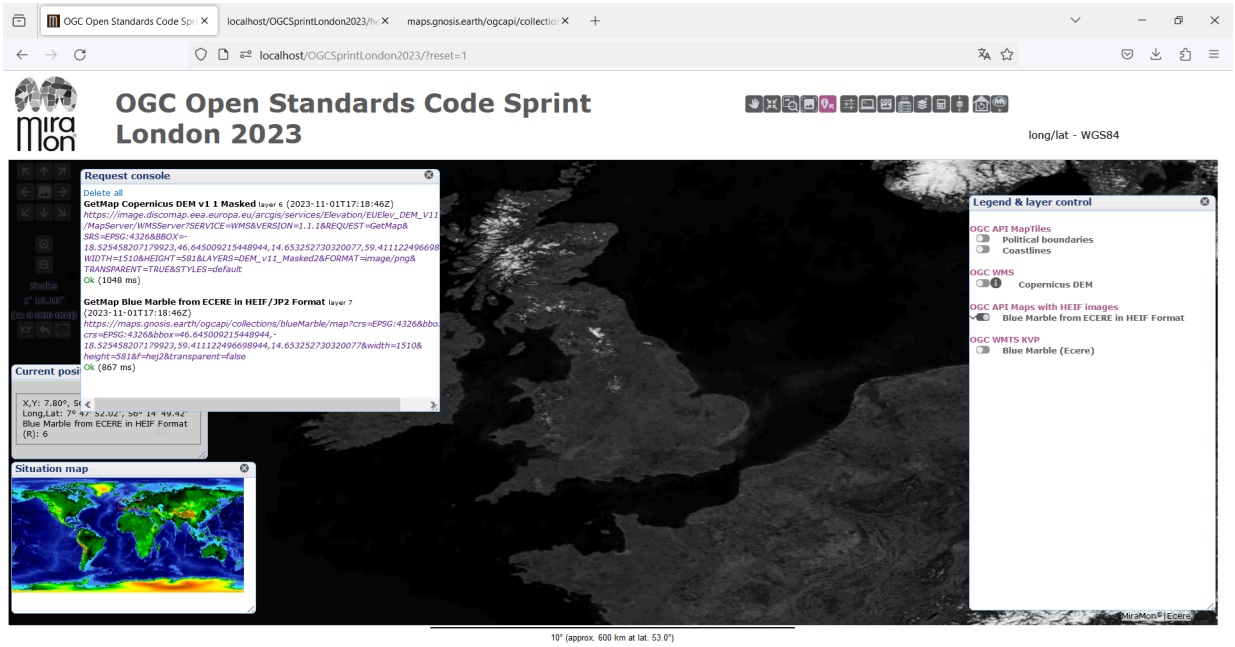


Figure 55 — Screenshot of the MiraMon Map Browser showing a HEIF image from an OGC API Maps.

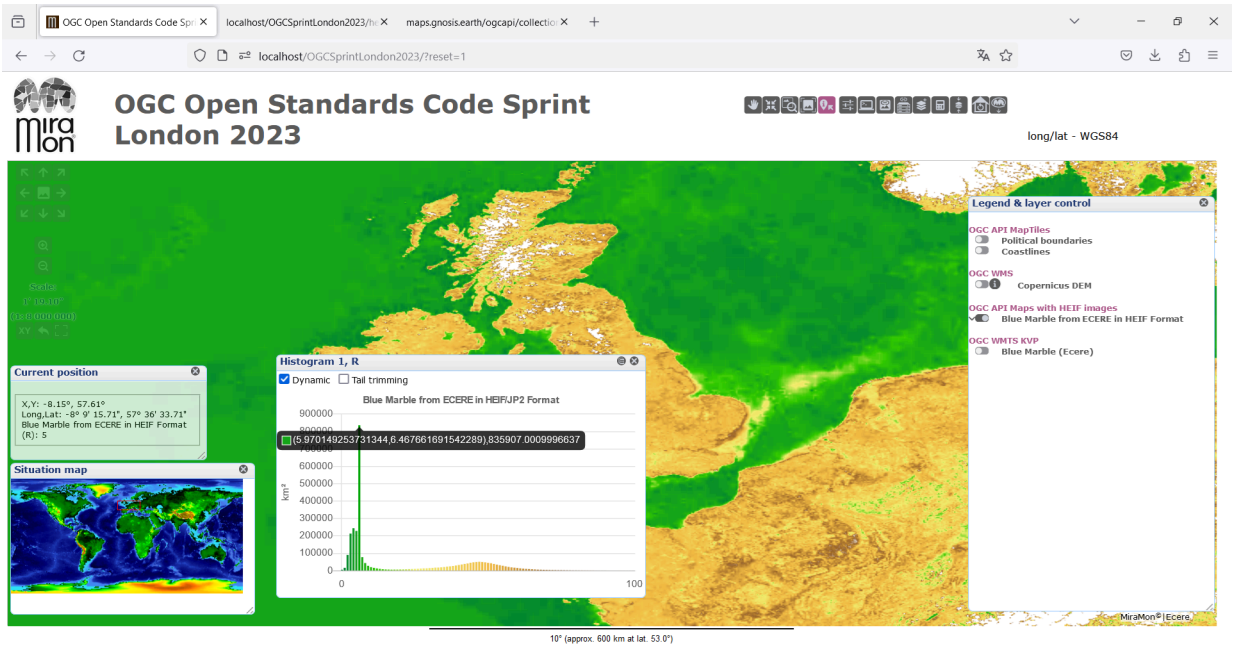


Figure 56 — Screenshot of the MiraMon Map Browser showing the histogram of a HEIF image from OGC API maps.

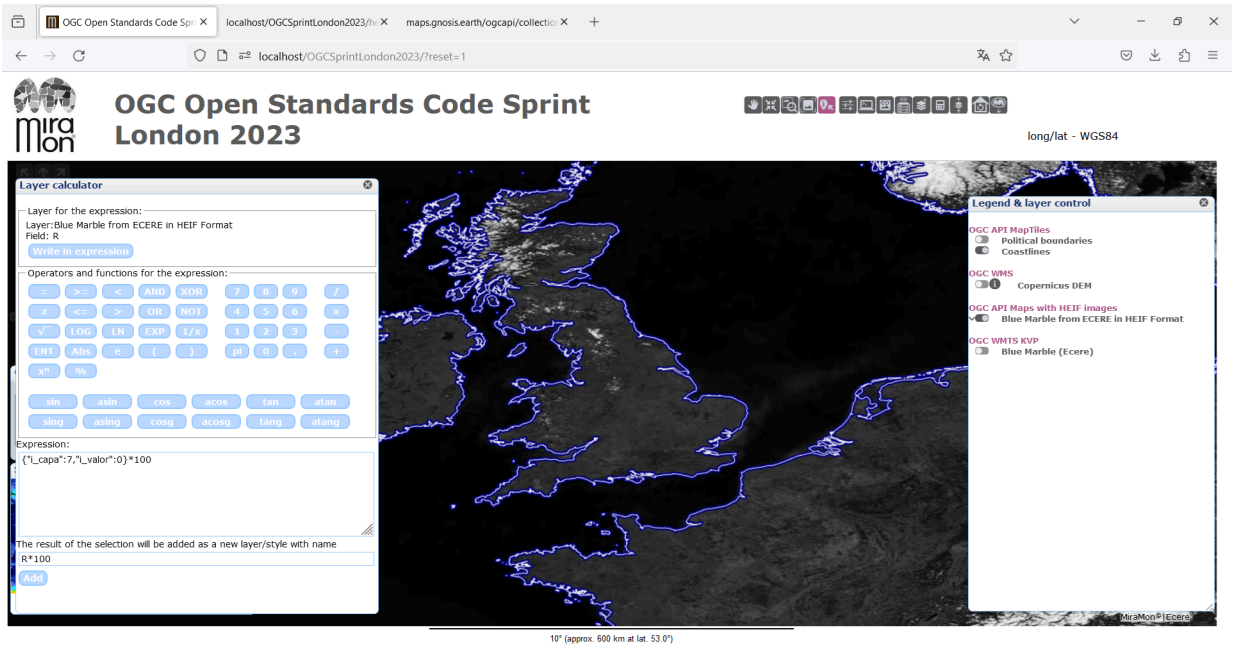


Figure 57 — Screenshot of the MiraMon Map Browser showing the calculator of layers with an HEIF image from OGC API maps.

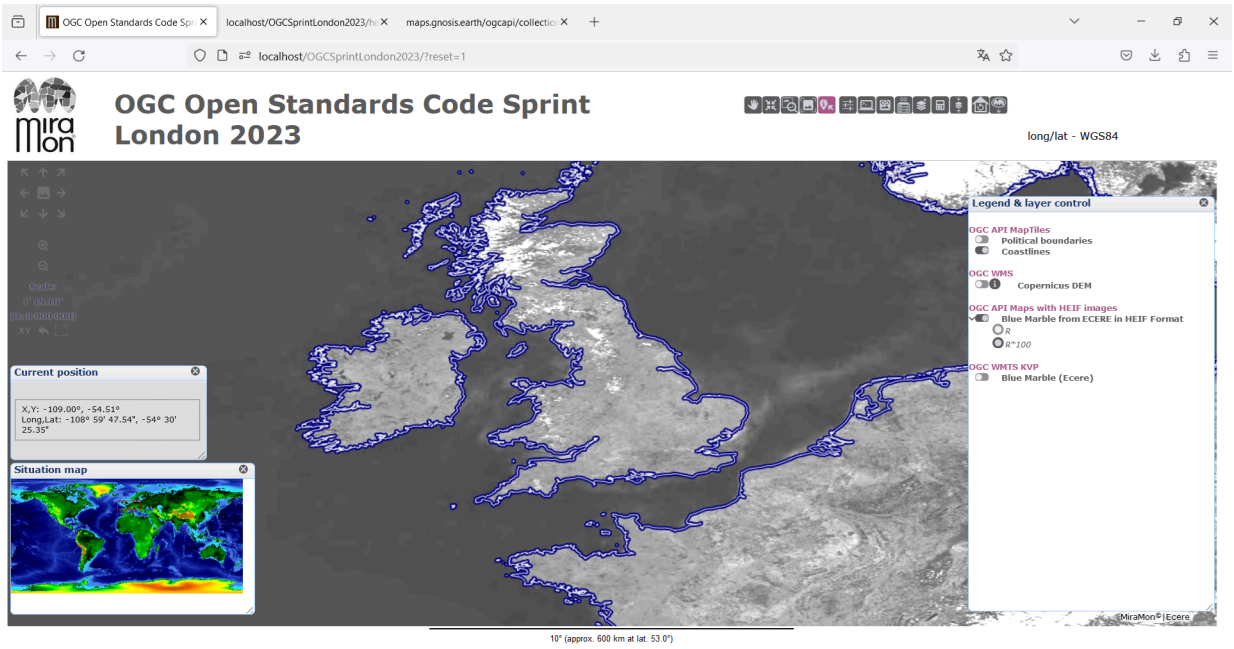


Figure 58 — Screenshot of the MiraMon Map Browser showing the result of a calculation using a HEIF image from OGC API maps.

5

DISCUSSION

5 DISCUSSION

5.1. JSON-FG

Several different types of applications were implemented or configured to support JSON-FG during the Code Sprint. Implementations included validators (linters), client-side applications, and server-side applications. The implementations and discussion led to a number of questions being logged on the SWG's GitHub repository (Issues 105 to 109). There were also some questions asked by developers that were getting started with JSON-FG during the code sprint. This suggests that the release of JSON-FG has to be accompanied by educational workshops and training to help the community adopt the standard. Messaging, especially in relation to the association with GeoJSON, will need to be clear.

One of the topics discussed by sprint participants was JSON-FG support for different coordinate reference systems. The participants also discussed the GeoJSON compatibility. Since the sprint took place in London, the British National Grid (EPSG:27700) was used as the coordinate reference system in example requests to an API implementing OGC API — Features. Clause 4.2.1 shows an example of how the British National Grid coordinate reference system can be used in JSON-FG documents. Several participants, for example those from MariaDB and interactive instruments, configured their applications to serve JSON-FG encoded data that was referenced to EPSG:27700. Support for the RD coordinate reference system that is used in the Netherlands was also demonstrated.

5.2. MUDDI

The work done by multiple sprint participants demonstrated that the MUDDI conceptual model could be used as the basis for implementing databases. MariaDB, Ordnance Survey, and Safe Software are some of the participants that implemented databases based on the MUDDI model. The draft JSON-FG and GML schemas of MUDDI that were provided by interactive instruments successfully enabled data to be served through an interface conforming to the OGC API — Features Standard. The same GML schemas were also used by Safe Software to generate and validate MUDDI GML. The analysis by IIT Bombay provided further insight into the similarity of feature properties in synthetic datasets versus those in the draft MUDDI schemas.

It is clear from the aforementioned implementations of MUDDI that MUDDI can be implemented. However, as the candidate Standard is that of a Conceptual Model, there is a need to specify a Standard for a logical model and data encoding. This need is evidenced by the observation that several developers had to make design-decisions (such as how to represent relationships between tables) regarding how to implement MUDDI during the code sprint (see Clause 4.2.4 for descriptions of some of those decisions). There were also some model interpretation decisions that had to be made as to what geometries should be used for each

feature type, prior interactive instruments generating the GML schemas. For the purposes of the pilot it was decided to select the simplest relevant geometries, such as points, lines, or polygons. An encoding standard would formalize some of those design-decisions so that the design-decisions become common to all implementations of MUDDI.

5.3. OGC API Standards

This section discusses findings that apply to multiple OGC API Standards, as well as implications of those findings.

5.3.1. Towards a common Publish/Subscribe Event Driven Approach

Activity during the code sprint included implementation of support for the Publish/Subscribe extension of OGC API — EDR. Through collaboration during the code sprint, the approach was also tested with OGC API — Features. As documented in Clause 4.3.1.2.4, initial discussion was started regarding the possibility of applying the approach in OGC API — Processes and is captured in a GitHub Issue. It is envisaged that common requirements can be identified and specified once the Publish/Subscribe approach has been tested with three or more OGC API Standards.

This code sprint had at least three different products implementing MQTT in support of Publish/Subscribe operations. It would be helpful for future code sprints to include other Event Driven architecture solutions such as Apache Kafka and RabbitMQ. Therefore, along with demonstrating that the Publish/Subscribe approach can support multiple OGC API Standards, the SWGs should endeavor to demonstrate support for different event notification tools in future code sprints.

5.3.2. Security

Multiple implementations of OGC API Standards demonstrated access control during the code sprint. For example, the Create, Replace, Update, Delete capabilities implemented by interactive instruments in ldproxy were protected through an access control layer. Another example was the CREAF and Secure Dimensions implementation of the STAplus extension of the OGC SensorThings API, described in Clause 4.1.5. The STAplus implementation was protected by an access control layer at the observation and feature level. These implementations, as well as others, demonstrate that OGC API Standards can support a variety of security models.

5.3.3. Schemas

A key aspect of the OGC API — Features work during the code sprint was in relation to the schema-focused part of the Standard (i.e., Part 5). The changes, which are listed in this GitHub Issue and described in Clause 4.1.3, point to the standard becoming more flexible in regard to data types of queryables and the logical model requirement becoming informative.

These changes potentially make the conformance class better able to support some of the requirements of other OGC API Standards that may need schema-support in future versions.

5.4. GIMI

The GIMI thread of the code sprint set out to accomplish two groups of tasks. The first group of tasks focused on prototyping an NGIIS Web API that:

- first, provided random access to GIMI content;
- second, extended the Web API to support storage of GIMI files; and
- third, extended the Web API to support updates to an existing GIMI file using random access.

As described in Clause 4.4.1 and Clause 4.4.6, the sprint participants were able to prototype capability that demonstrated how implementations of OGC API — Tiles and OGC API — Coverages could provide random access to content from GIMI files. The ability to support storage of GIMI files through an implementation of OGC API — Processes was prototyped after the three-day code sprint and is documented in Clause 4.4.2. The ability to update a GIMI file through an OGC API was not demonstrated in this code sprint due to time constraints and limitations of some of the key HEIF libraries. The sprint participants were able to record five issues in libheif, some of which have since been fixed by the maintainers of libheif. The sprint participants were also able to propose changes to mp4box.js and misb.js through pull requests.

The second group of tasks focused on client applications:

- first, implementation of a client that can generate GIMI files with different types of imagery and store them using the NGIIS Web API; and
- second, implementation of client applications that can access GIMI data, show/display content id, timing, and security information.

As described in Clause 4.4.3, a custom application with a variety of libraries was used in the generation of GIMI files. The resulting GIMI files used HEVC (H.265), JPEG 2000, and uncompressed encodings. Additional experimentation was conducted as documented in Clause 4.4.4. Two web-browser based applications were configured to read and display GIMI files, namely MiraMon (described in Clause 4.4.7) and the web-gimi-viewer (described in Clause 4.4.5). In addition to the the web-browser based applications, a plug-in for the QGIS Desktop application was implemented for reading GIMI files as described in Clause 4.4.3.

6

CONCLUSIONS

6 CONCLUSIONS

Over a period of three days, the code sprint provided an environment for the development and testing of prototype implementations of open standards. The standards included implementations of draft and approved OGC Standards, as well as the draft GIMI specification and AsyncAPI. The in-person participation of several editors of OGC Standards meant that developers could provide feedback to the editors almost instantly. Furthermore, the collaborative environment enabled developers to fix open issues in products, identify new issues, develop new product features, improve documentation, and test interoperability with 3rd party products.

Regarding JSON-FG, the engineering report concludes that due to the evident increase in adoption across the developer community, there is an increasing need to offer validators so that developers can check for compliance more easily. Such validators may include unofficial utilities, in addition to the official executable test suite that will be provided by the OGC Compliance Program.

In relation to MUDDI, the engineering report concludes that the MUDDI Conceptual Model is implementable and could easily be used as the basis for a logical model that can be encoded in a variety of formats. Once the Conceptual Model has been approved as an OGC Standard, the SWG should therefore consider developing standards for encoding MUDDI-conformant data in GeoPackage, GML, or JSON-FG.

In relation to CityGML version 3, the engineering report concludes that there are clear pathways for generating CityGML v3 data based on CityGML v2 and also from other related standards such as IFC.

The conclusions regarding the emerging extensions to OGC API Standards can be summarized as follows: the PubSub, Security, and Schemas capabilities examined in this code sprint could potentially support other OGC API Standards. Therefore, there is a need for the various OGC API SWGs to consider the potential application of the extensions to the standards within the remit of the SWGs.

Regarding GIMI, the engineering report concludes that with minor changes to popular open-source base libraries, a wide range of software can quickly make use of GIMI capabilities including software that implements OGC API Standards. However, to fully leverage the capabilities of the GIMI specification, the base libraries will need to include the required functionality at production.

Overall, this engineering report concludes that the code sprint met all of its objectives and achieved its goal of accelerating the development and implementation of OGC Standards in software products. The following subsection presents recommendations for future work.

6.1. Future Work

6.1.1. OGC API Standards

The sprint participants made the following recommendations for future work items on OGC API Standards.

- Extension of OGC API — Records to support faceting.
- Use case experimentation to examine how the draft PubSub extension of OGC API Standards can support various OGC API Standards workflows:
    - OGC API — Records use cases could include aggregation/metadata or ingest/publication;
    - OGC API — Processes use cases could include subscribing to async jobs channels; and
    - OGC API — Common — Part 2 use cases could include publishing of data updates (e.g., publication of new NWP model run for EDR or Maps, or associating OGC API collections from AsyncAPI (x-ogc-api-collectionId)
- Prototyping and specification on an OpenAPI definition document for SensorThings API.
- Additional support for multiband raster data in OGC API — Maps/Tiles/Coverages (e.g., multispectral imagery, etc) which could include drafting of a Best Practice of when to use OGC API — Maps/Tiles/Coverages.

6.1.2. OGC Encoding Standards

The sprint participants made the following recommendations for future work items on OGC Encoding Standards.

- Tiled complex, signed, and floating point data in multiband raster data which could include experimentation where the tiles provide a way for machines to process an area of interest (potentially in parallel), rather than as pictures to show to the user.
- Performance benchmarking between GIMI, GeoTIFF, COG, JPEG 2000 and other imagery formats for a variety of use cases.
- Extraction of a conceptual model from GMLJP2 to make it applicable to any ISOBMFF profile and encodable in different formats.
- Development of an Implementation Standard for the MUDDI logical model which could be designed as a multi-part standard with separate parts for JSON-FG, GML, and GeoPackage.

- Future work on the storage aspects of the MUDDI task could involve using the MUDDI logical schema as a template for MUDDI compliance.

6.1.3. GIMI

The sprint participants made the following recommendations for future work items on GIMI.

- Extension of open-source base libraries used by several other products (e.g., libheif and GDAL) for creation and modification of GIMI files.
- Prototype streaming of large images from client application to server for storage through OGC API — Processes.
- Prototyping of a JSON alternative to KLV metadata (e.g., a future ontology for GIMI could include the same information that is in the KLV bytestream and have it encoded in JSON-LD).

A

ANNEX A (INFORMATIVE) REVISION HISTORY

A ANNEX A (INFORMATIVE) REVISION HISTORY

Table — Revision History

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2023-10-20	0.1	G. Hobona	all	initial version
2023-11-20	0.2	G. Hobona	all	submitted to TC

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Gobe Hobona, Joana Simoes, Tom Kralidis, Martin Desruisseaux, Angelos Tzotsos: OGC 23-025, *2023 Open Standards and Open Source Software Code Sprint Summary Engineering Report*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/PER/ogc-osgeo-asf-codesprint2023.

[2] Carl Reed: OGC 15-113r6, *Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure*. Open Geospatial Consortium (2021). http://www.opengis.net/doc/IS/CDB-core/1.2.0.

[3] Mark Burgoyne, David Blodgett, Charles Heazel, Chris Little: OGC 19-086r6, *OGC API — Environmental Data Retrieval Standard*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/IS/ogcapi-edr-1/1.1.0.

[4] Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r4, *OGC API — Features — Part 1: Core corrigendum*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-features-1/1.0.1.

[5] Clemens Portele, Panagiotis (Peter) A. Vretanos: OGC 18-058r1, *OGC API — Features — Part 2: Coordinate Reference Systems by Reference corrigendum*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-features-2/1.0.1.

[6] Benjamin Pross, Panagiotis (Peter) A. Vretanos: OGC 18-062r2, *OGC API — Processes — Part 1: Core*. Open Geospatial Consortium (2021). http://www.opengis.net/doc/IS/ogcapi-processes-1/1.0.0.

[7] Joan Masó, Jérôme Jacovella-St-Louis: OGC 20-057, *OGC API — Tiles — Part 1: Core*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-tiles-1/1.0.0.

[8] Tatjana Kutzner, Carl Stephen Smyth, Claus Nagel, Volker Coors, Diego Vinasco-Alvarez, Nobuhiro Ishi: OGC 21-006r2, *OGC City Geography Markup Language (CityGML) Part 2: GML Encoding Standard*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/IS/CityGML-2/3.0.0.

[9] Andreas Matheus: OGC 22-022r1, *OGC SensorThings API Extension: STAplus 1.0*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/is/sensorthings-staplus/1.0.0.

[10] Steve Liang, Tania Khalafbeigi, Hylke van der Schaaf: OGC 18-088, *OGC SensorThings API Part 1: Sensing Version 1.1*. Open Geospatial Consortium (2021). http://www.opengis.net/doc/is/sensorthings/1.1.0.

[11] Lucio Colaiacomo, Joan Masó, Emmanuel Devys, Eric Hirschorn: OGC 08-085r8, *OGC® GML in JPEG 2000 (GMLJP2) Encoding Standard*. Open Geospatial Consortium (2018). http://www.opengis.net/doc/IS/GMLJP2/2.1.0.

[12] Jeff Yutzler: OGC 12-128r18, *OGC® GeoPackage Encoding Standard*. Open Geospatial Consortium (2021). http://www.opengis.net/doc/IS/geopackage/1.3.1.

[13] Clemens Portele, Panagiotis (Peter) A. Vretanos: OGC Features and Geometries JSON — Part 1: Core draft, Open Geospatial Consortium. https://docs.ogc.org/DRAFTS/21-045.html

[14] Alan Leidner, Andrew Hughes, Carsten Roensdorf, Neil Brammall, Liesbeth Rombouts, Joshua Lieberman: OGC MUDDI Conceptual Model draft, Open Geospatial Consortium. https://portal.ogc.org/files/106383

[15] Tom Kralidis, Mark Burgoyne, Steve Olson, Shane Mill: OGC API — Environmental Data Retrieval — Part 2: Publish-Subscribe workflow, (draft), Open Geospatial Consortium. https://github.com/opengeospatial/ogcapi-environmental-data-retrieval/tree/master/extensions/pubsub/standard

[16] OGC API — Coverages — Part 1: Core (draft), Open Geospatial Consortium. https://docs.ogc.org/DRAFTS/19-087.html

[17] OGC API — Maps — Part 1: Core (draft), Open Geospatial Consortium. https://docs.ogc.org/DRAFTS/20-058.html

[18] OGC API — Records — Part 1: Core (draft), Open Geospatial Consortium. https://docs.ogc.org/DRAFTS/20-004.html

[19] GEOINT Imagery Media for ISR (GIMI) Profile of ISOBMFF (draft 0.4)

[20] Linda van den Brink, Arnoud de Boer: Testing OGC JSON-FG Draft Specification and its GDAL-DEV Implementation. Geonovum (2023). https://geonovum-labs.github.io/test-ogc-json-fg

[21] Neil Brammall: Going underground: developing and testing an international standard for subsurface data. Geospatial Commission (2023). https://geospatialcommission.blog.gov.uk/2023/12/19/going-underground-developing-and-testing-an-international-standard-for-subsurface-data/