OGC® DOCUMENT: 23-025

External identifier of this OGC® document: http://www.opengis.net/doc/PER/ogc-osgeo-asf-codesprint2023

Open Geospatial Consortium

2023 OPEN STANDARDS AND OPEN SOURCE SOFTWARE CODE SPRINT SUMMARY ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2023-05-12
Approval Date: 2023-06-08
Publication Date: 2023-11-01
Editor: Gobe Hobona, Joana Simoes, Tom Kralidis, Martin Desruisseaux, Angelos Tzotsos

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 vii

II. KEYWORDS viii

III. SUBMITTERS viii

IV. ABSTRACT ix

1. SCOPE 2

2. NORMATIVE REFERENCES 4

3. TERMS, DEFINITIONS AND ABBREVIATED TERMS 6
3.5. Abbreviated terms 7

4. HIGH-LEVEL ARCHITECTURE 9
4.1. Approved OGC Standards 9
4.1.1. OGC SensorThings API 9
4.1.2. OGC API — Features 10
4.1.3. OGC Styles and Symbology 10
4.1.4. OGC GeoXACML 10
4.1.5. OGC GeoAPI 11
4.2. Candidate OGC Standards 11
4.2.1. OGC GeoPose 11
4.2.2. OGC API — Records 12
4.2.3. OGC GeoParquet 12
4.3. Other Specifications 12
4.3.1. AsyncAPI 12
4.3.2. STAC API 12
4.4. ASF Apache Projects 13
4.4.1. Apache Arrow 13
4.4.2. Apache Baremaps 13
4.4.3. Apache Sedona 13
4.4.4. Apache SIS 13
4.5. OSGeo Projects 14
4.5.1. OSGeo GeoNetwork 14
4.5.2. OSGeo GeoServer 14
4.5.3. OSGeo OpenLayers 14
4.5.4. OSGeo OWSLib 14
4.5.5. OSGeo pycsw 14

4.5.6. OSGeo pygeoapi 15
4.5.7. OSGeo QGIS 15
4.6. Community Open Source Projects 15
4.6.1. OSGeo ZOO-Project 15
4.6.2. TEAM Engine 15
4.6.3. CesiumJS 16
4.6.4. OSGeo Leaflet 16
4.6.5. Camptocamp ogc-client 16
4.6.6. HomeAssistant-SensorThings 16
4.6.7. Maplibre 16
4.6.8. OSGeo MDME 16
4.6.9. OL-Cesium 17
4.6.10. OSGeo pygeometa 17
4.6.11. Geomapfish 17
4.6.12. Smapshot 17
4.6.13. go-ogc 18
4.6.14. GPQ 18
4.6.15. ldproxy 18

5. RESULTS 20
5.1. Approved and Candidate OGC Standards 20
5.1.1. CQL2 20
5.1.2. OGC API — Tiles 21
5.1.3. OGC Styles and Symbology 21
5.1.4. OGC Features and Geometries JSON (JSON-FG) 21
5.1.5. OGC GeoXACML 22
5.1.6. OGC GeoAPI 29
5.1.7. OGC API — Records 30
5.1.8. OGC API — Features 30
5.1.9. OGC API — Environmental Data Retrieval 32
5.2. Other Specifications 33
5.2.1. AsyncAPI 33
5.3. ASF Apache Projects 35
5.3.1. Apache Baremaps 35
5.3.2. Apache Sedona 35
5.3.3. Apache SIS 35
5.4. OSGeo Projects 36
5.4.1. GeoNetwork 36
5.4.2. OSGeo GeoServer 36
5.4.3. OSGeo OpenLayers 37
5.4.4. OWSLib 37
5.4.5. OSGeo pycsw 37
5.4.6. OSGeo pygeoapi 37
5.4.7. OSGeo QGIS 39
5.5. Community Open Source Projects 40
5.5.1. OSGeo ZOO-Project 40
5.5.2. TEAM Engine 40

5.5.3. HomeAssistant-SensorThings 41
5.5.4. Maplibre 43
5.5.5. MDME 44
5.5.6. OL-Cesium 44
5.5.7. OSGeo pygeometa 44
5.5.8. Smapshot 44
5.5.9. go-ogc 46
5.5.10. GPQ 47

6. DISCUSSION 49
6.1. IOGP GIGS tests and the OGC GeoAPI Standard 49
6.2. Home Assistant and OGC SensorThings API 50
6.3. IndoorGML 51
6.4. GeoXAML 3.0 51
6.5. pygeoapi 52
6.6. Styling in OpenLayers 56
6.7. go-ogc 58

7. CONCLUSIONS 60
7.1. Future Work 60

ANNEX A (INFORMATIVE) REVISION HISTORY 62

BIBLIOGRAPHY 64

LIST OF FIGURES

Figure 1 — High Level Overview of the Sprint Architecture 9
Figure 2 — XACML Flow Diagram 23
Figure 3 — Left: Feature type “poly_landmarks” without PEP → Central Park feature(s) are included; Right: Feature type “poly_landmarks” with PEP → Central Park feature(s) are excluded! 26
Figure 4 — Screenshot of a pygeoapi landing page 33
Figure 5 — Screenshot of the pygeoapi OGC API - Features provider for the ERDDAP data server. 38
Figure 6 — pygeoapi OGC API - Pub/Sub architecture. 38
Figure 7 — pygeoapi OGC API - Pub/Sub architecture. 39
Figure 8 — Partial screenshot of a Home Assistant dashboard showing SensorThings observations from the British Geological Survey, from their FROST server. 41
Figure 9 — Screenshot of Maplibre demonstration 43
Figure 10 — A screenshot of Smapshot 45
Figure 11 — Position and orientation from a geopose represented in Smapshot 45
Figure 12 — Example execution of the GIGS test sofware 49

Figure 13 — Differences in Units of measure representation in a sample of SensorThings API instances 50
Figure 14 — Screenshot of a rendered HTML-encoded pygeoapi asyncapi document 53
Figure 15 — Screenshot of MQTT Explorer receiving notifications from an OGC API - Features implementation 55
Figure 16 — Screenshot of a WIS 2.0 prototype receiving notifications through a Pub/Sub mechanism 56
Figure 17 — A screenshot of an OpenLayers application that supports client-side modification of styles 57
Figure 18 — Screenshot of the Planet Catalog interface 58

I EXECUTIVE SUMMARY

The mutually beneficial relationship between open standards and open source software has been instrumental in the uptake of geospatial technologies across a wide variety of industries. Whereas open source software has enabled many organizations to rapidly implement open standards, open standards have enabled open source software to support a multitude of use cases through efficient integration and greater interoperability. An enduring challenge, however, has been how and when to bring developers from the open standards and open source software communities together.

From the 25th to the 27th of April 2023, the Open Geospatial Consortium (OGC), the Apache Software Foundation (ASF), and the Open Source Geospatial Foundation (OSGeo) held their third joint Open Standards and Open Source Software Code Sprint. The code sprint served to accelerate the support of open geospatial standards within the software developer community.

The code sprint brought together developers of Open Standards, Open Source Software, and Proprietary Software into the same physical space. Such face-to-face collaboration had been made possible by the support of the Ordnance Survey, the event's Gold-level Sponsor, and the support of Camptocamp, the event host, as well as co-sponsors of the catering namely HEIG-VD (School of Engineering and Management), EPFL, University of Lausanne, State of Neuchâtel, State of Vaud, and Camptocamp.

Previous code sprints jointly organized by OGC, ASF, and OSGeo had been completely virtual events due to the pandemic. Therefore, this third code sprint held special significance in that it was the first hybrid event (supporting both face-to-face and remote participation) that had been jointly organized by the three organizations. This Engineering Report (ER) summarizes the main achievements of the code sprint.

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

The Open Source Geospatial Foundation (OSGeo) is a not-for-profit organization whose mission is to foster global adoption of open geospatial technology by being an inclusive software foundation devoted to an open philosophy and participatory community driven development. The foundation consists of projects that develop open source software products.

The Apache Software Foundation (ASF) is an all-volunteer community comprising 815 individual Members and 8,500 Committers on six continents stewarding more than 227 million lines of code and overseeing more than 350 Apache projects and their communities.

The code sprint provided an environment for development and testing of prototype implementations of open standards, including implementations of draft and approved OGC Standards. Further, the code sprint provided a collaborative environment for developers to fix open issues in products, develop new features, improve documentation, improve interoperability with other libraries/products, and develop prototype implementations of OGC Standards. In addition, the code sprint enabled the participating developers to provide feedback to the editors

of OGC Standards. The code sprint therefore met all of its objectives and achieved its goal of accelerating the support of open geospatial standards within the developer community.

The sprint participants made the following recommendations for future work:

- experimentation and prototyping of implementations of OGC Styles and Symbology;
- provide examples of the use of OGC Rainbow (formerly known as the OGC Definitions Server) to support the representation of units of measure in sensor-related standards;
- experimentation and prototyping of implementations of OGC GeoXACML;
- advance the development of an Executable Test Suite for OGC API — Tiles;
- advance the development of an Executable Test Suite for OGC API — Records; and
- advance the development of an Executable Test Suite for OGC GeoXACML.

II KEYWORDS

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

OGC, OSGeo, ASF, Apache, hackathon, code sprint, standards, geospatial, API, open source

III SUBMITTERS

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

NAME	ORGANIZATION	ROLE
Gobe Hobona	Open Geospatial Consortium	Editor
Joana Simoes	Open Geospatial Consortium	Editor
Angelos Tzotsos	Open Source Geospatial Foundation	Editor
Martin Desruisseaux	Geomatys	Editor
Tom Kralidis	Meteorological Service of Canada	Editor

NAME	ORGANIZATION	ROLE
Andreas Matheus	Secure Dimensions	Contributor
Olivia Guyot	Camptocamp	Contributor
Tim Schaub	Planet	Contributor
Dirk Stenger	lat/lon	Contributor
Jody Garnett	OSGeo	Contributor
Stéphane Lecorney	University of Applied Sciences Western Switzerland (HEIG-VD)	Contributor
Paul van Genuchten	ISRIC- World soil information	Contributor
Iván Sánchez Ortega	OSGeo	Contributor
Florent Gravin	Camptocamp	Contributor
Elena Robu	Astun Techonology Ltd	Contributor
Priyanka Talwar	IIT Bombay	Contributor
Shweta Naik	IIT Bombay	Contributor
Clemens Portele	interactive instruments GmbH	Contributor
Jérôme Jacovella-St-Louis	Ecere Corporation	Contributor

IV ABSTRACT

The subject of this Engineering Report (ER) is a code sprint that was held from the 25th to the 27th of April 2023 to advance support of open geospatial standards within the developer community, while also advancing the standards themselves. The code sprint was organized by the Open Geospatial Consortium (OGC), the Open Source Geospatial Foundation (OSGeo), and the Apache Software Foundation (ASF). The code sprint was sponsored by the Ordnance Survey and hosted by Camptocamp.

1 SCOPE

1 SCOPE

This Engineering Report (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.

2

NORMATIVE REFERENCES

2 NORMATIVE REFERENCES

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

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

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

W3C: HTML5, W3C Recommendation, https://www.w3.org/TR/html5/

Schema.org: https://schema.org/docs/schemas.html

Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r4, *OGC API — Features — Part 1: Core corrigendum*. Open Geospatial Consortium (2022). https://docs.ogc.org/is/17-069r4/17-069r4.html.

Mark Burgoyne, David Blodgett, Charles Heazel, Chris Little: OGC 19-086r5, *OGC API — Environmental Data Retrieval Standard*. Open Geospatial Consortium (2022). https://docs.ogc.org/is/19-086r5/19-086r5.html.

Benjamin Pross, Panagiotis (Peter) A. Vretanos: OGC 18-062r2, *OGC API — Processes — Part 1: Core*. Open Geospatial Consortium (2021). https://docs.ogc.org/is/18-062r2/18-062r2.html.

Joan Masó, Jérôme Jacovella-St-Louis: OGC 20-057, *OGC API — Tiles — Part 1: Core*. Open Geospatial Consortium (2022). https://docs.ogc.org/is/20-057/20-057.html.

Charles Heazel: OGC 19-072, *OGC API — Common — Part 1: Core*. Open Geospatial Consortium (2023). https://docs.ogc.org/is/19-072/19-072.html.

3

TERMS, DEFINITIONS AND ABBREVIATED TERMS

3 TERMS, DEFINITIONS AND ABBREVIATED TERMS

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

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

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

3.1. API

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

3.2. coordinate reference system

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

3.3. OpenAPI Document

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

3.4. Web API

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

3.5. Abbreviated terms

API	Application Programming Interface
ASF	Apache Software Foundation
CITE	Compliance Interoperability & Testing Evaluation
CRS	Coordinate Reference System
CSW	Catalogue Services for the Web
EDR	Environmental Data Retrieval
GIS	Geographic Information System
MOU	Memorandum of Understanding
OGC	Open Geospatial Consortium
OSGeo	Open Source Geospatial Foundation
OWS	OGC Web Services
REST	Representational State Transfer
SDI	Spatial Data Infrastructure
TEAM	Test, Evaluation, And Measurement Engine
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WMTS	Web Map Tile Service
WPS	Web Processing Service

4

HIGH-LEVEL ARCHITECTURE

4 HIGH-LEVEL ARCHITECTURE

The focus of the sprint was on the support of implementations of open geospatial standards across various open source software projects. Implementations of approved and candidate OGC Standards were deployed in participants' own infrastructure in order to build a solution 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 open geospatial standards. The architecture also included several different software libraries that support open geospatial standards and enable the extraction, transformation, and loading of geospatial data.

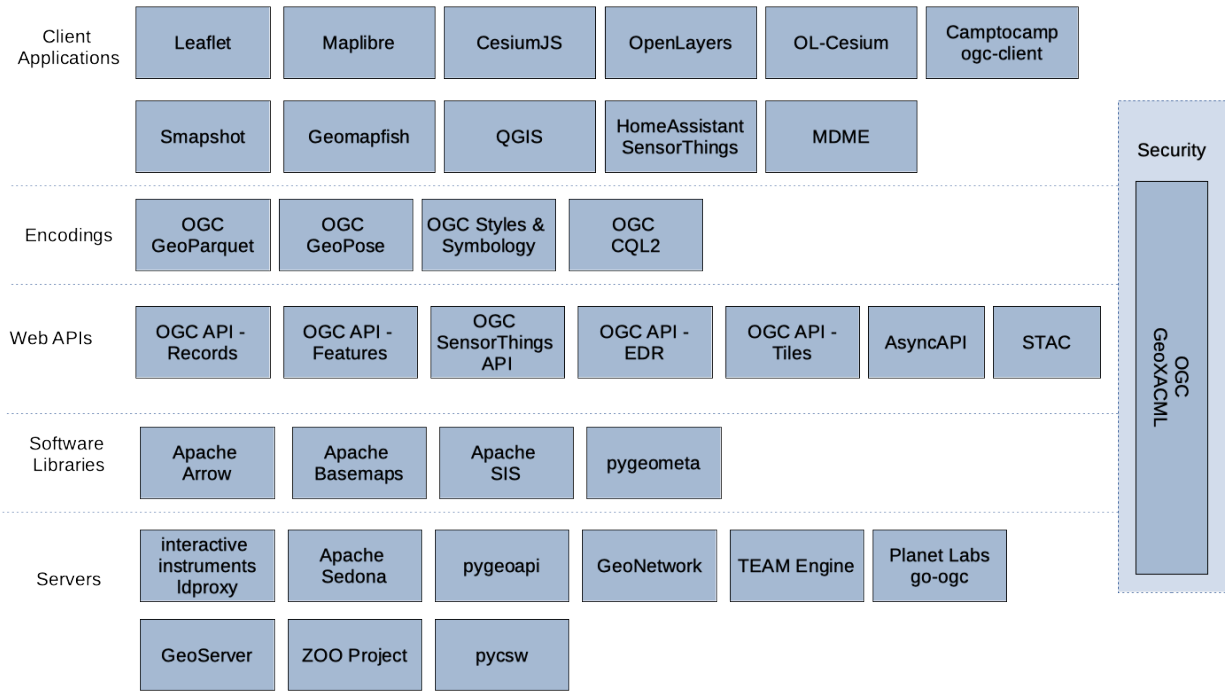

Figure 1 — High Level Overview of the Sprint Architecture

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

4.1. Approved OGC Standards

4.1.1. OGC SensorThings API

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

variety of IoT sensor systems. The Tasking part provides a standard way for tasking IoT devices, such as sensors or actuators. The SensorThings API follows REST principles and uses JSON for encoding messages, as well as MQTT for publish/subscribe operations.

4.1.2. OGC API — Features

The OGC API — Features Standard offers the capability to create, modify, and query spatial data on the Web. The Standard specifies requirements and recommendations for APIs that want to follow a standard way of sharing 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). Part 2 enables the use of different CRS, in addition to the WGS 84. 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.

The OGC API — Features Standard is part of the OGC API family of standards. OGC API Standards define modular API building blocks to spatially enable Web APIs in a consistent way. The standards make use of the OpenAPI specification for defining the API building blocks.

4.1.3. OGC Styles and Symbology

The OGC Symbology Conceptual Model: Core Part Standard (OGC 18-067r3), also known as OGC SymCore, specifies the conceptual basis to define symbology rules for the portrayal of geographical data. It is modular and extensible (one core model, many extensions), also encoding agnostic (one symbology model, many encodings). It contains a minimal set of abstract classes representing explicit extension points of the model.

OGC Styles and Symbology is a vision for a standard that implements the OGC Symbology Conceptual Model: Core Part Standard (OGC 18-067r3). The Web Mapping Code Sprint was a key moment to consolidate the roadmap to achieve this vision. The plan towards such a candidate SymCore 2.0 Standard consists of the definition of a conceptual and logical model, Cascading Style Sheets (CSS) and JavaScript Object Notation (JSON) encodings, as well as a mapping to SLD/SE (eventually with existing well known vendor options). The requirements described will be supported by illustrated and encoded cartographic use cases.

4.1.4. OGC GeoXACML

The OGC GeoXACML Standard defines a geo-specific extension to the XACML Policy Language as defined by the OASIS standard “eXtensible Access Control Markup Language (XACML) that allows managing access to geographic information and services in an interoperable way across jurisdictions. Although XACML is meant to be used as a multi-purpose language, XACML does not have the capabilities to express geo-specific constraints on access rights. GeoXACML provides support for spatial data types and spatial authorization decision functions. Those data types and functions can be used to define additional spatial constraints for XACML based

policies. GeoXACML defines extensions for the policy encoding schema only and thus does not affect the XACML context schema.

Implementations of OGC API Standards are designed to work over HTTP(S). Securing the deployed APIs is not of concern to the OGC API Standards. However, the requirement to describe the API's endpoint via OpenAPI introduces standard options to enable authentication. Knowing the user that accesses an API is an important aspect for many use cases. Also, user identification is often used to control access to the data, served by the API. However, OGC API Standards do not currently provide guidance on how to apply access control but instead delegate the responsibility of describing the security scheme to the OpenAPI definition document. This leads to an interoperability issue if access is denied but there is no common (standardized) way to "tell the reason". OGC's GeoXACML 3.0 (currently still draft) Standard supports the interoperable exchange of access conditions as policies. GeoXACML 3.0 defines the geospatial extension to OASIS XACML 3.0 and thereby extends the ability to include Attribute Based Access Control (ABAC) also on geometries.

The challenge for the GeoXACML part of the code sprint was to develop a Policy Enforcement Point (PEP) that intercepts Web Feature Service (WFS) 2.0 requests and modifies the request according to an access decision returned by a GeoXACML 3.0 policy. The access control use case was the following: "WFS GetFeature requests shall exclude the NYC Central Park for feature type poly_landmarks".

4.1.5. OGC GeoAPI

OGC GeoAPI 3.0.2 is an OGC Standard providing a set of programming interfaces for geospatial applications. The GeoAPI interfaces closely follow the abstract models published by OGC and ISO specifications. The current release covers interfaces for metadata handling and for geodetic referencing (map projections). The GeoAPI interfaces provide a layer which separates client code, which call the API from library code, which implements the API. This follows a similar pattern to the well known JDBC or ODBC API which provides a standardized interface to databases. Clients can use the API without concern for the particular implementation which they will use.

4.2. Candidate OGC Standards

4.2.1. OGC GeoPose

OGC GeoPose is an OGC Draft Standard for exchanging the location and orientation of real or virtual geometric objects (Poses) within reference frames anchored to the earth's surface (Geo) or within other astronomical coordinate systems. GeoPose specifies a single encoding in JSON format (IETF RFC 8259). The use cases addressed by GeoPose involve interactions between information systems or between an information system and a storage medium. The role of a GeoPose is to convey the orientation and position of a real or virtual object.

4.2.2. OGC API — Records

OGC API — Records provides discovery and access to metadata records about resources such as features, coverages, tiles / maps, models, assets, services or widgets. The draft specification enables the discovery of geospatial resources by standardizing the way collections of descriptive information about the resources (metadata) are exposed. The draft specification 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. The draft OGC API – Records specification is part of the OGC API family of standards.

4.2.3. OGC GeoParquet

The OGC GeoParquet candidate Standard defines how geospatial data should be stored in Apache Parquet format including the representation of geometries and the required additional metadata. The Apache Parquet project provides an open source columnar file format that supports both data storage and retrieval.

4.3. Other Specifications

4.3.1. 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.

4.3.2. STAC API

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

4.4. ASF Apache Projects

4.4.1. Apache Arrow

Apache Arrow is a software framework for developing applications that can process columnar data in data analytics solutions. The software framework offers a column-oriented memory format that can represent flat and hierarchical data for analytic operations on modern CPU and GPU hardware. Apache Arrow supports the Apache Parquet Format, which is the format on which the OGC GeoParquet candidate standard is being built to extend.

4.4.2. Apache Baremaps

Apache Baremaps is an open source software toolkit and a set of infrastructure components for creating, operating, and publishing maps online. The toolkit provides a data pipeline that enables developers to build maps from a variety of data sources. The toolkit enables cartographers to customize the content and the style of a map as well as supporting services, such as location search and IP to location.

4.4.3. Apache Sedona

Apache Sedona is a high-performance cluster computing system designed specifically for processing large-scale spatial data. The system leverages a number of open-source libraries to execute spatial operations such as Coordinate Reference System (CRS) transformation and file reading.

4.4.4. Apache SIS

Apache Spatial Information System (SIS) is a free and open source software library for developing geospatial applications. The library is an implementation of OGC GeoAPI 3.0.1 interfaces and can be used for desktop or server applications. Services provided by Apache SIS include metadata, coordinate transformations, filtering, and grid coverages. The library is implemented using the Java programming language.

In this code sprint Apache SIS was used for testing the Geospatial Integrity of Geoscience Software (GIGS) test runners.

4.5. OSGeo Projects

4.5.1. OSGeo GeoNetwork

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

4.5.2. OSGeo GeoServer

GeoServer is a Java-based software server that allows users to view and edit geospatial data. Using open standards by the OGC, GeoServer allows for flexibility in map creation and data sharing.

4.5.3. OSGeo OpenLayers

OpenLayers is a library for developing browser-based mapping applications. It works with vector and raster data from a variety of sources and is able to reproject data for rendering. The library has broad support for OGC protocols and formats, including Web Map Service (WMS), Web Map Tile Service (WMTS), Web Feature Service (WFS), Well Known Text (WKT), Well Known Binary (WKB), Geography Markup Language (GML), GeoTIFF/COG, and KML. In addition, OpenLayers has support for community specifications such as GeoJSON, XYZ tiles, TileJSON, and more. OpenLayers is written in JavaScript and is available under the BSD 2-Clause license.

The `ol/source/OGCMapTile` module provides a source for map tiles from an OGC API – Tiles implementation. The `ol/source/OGCVectorTile` module provides a source for tiled vector feature data from an OGC API — Tiles implementation.

4.5.4. OSGeo OWSLib

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

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

4.5.5. OSGeo pycsw

pycsw is an implementation of OGC API — Records and the OGC's Catalogue Service for the Web (CSW) written in Python. Started in 2010 (more formally announced in 2011), pycsw allows for the publishing and discovery of geospatial metadata via numerous APIs (CSW 2/CSW 3, OpenSearch, OAI-PMH, SRU), providing a standards-based metadata and catalogue component of spatial data infrastructures. pycsw is Open Source, released under an MIT license, and runs on

all major platforms (Windows, Linux, Mac OS X). pycsw is an official OSGeo Project as well as an OGC Reference Implementation.

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

4.5.6. OSGeo pygeoapi

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

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

4.5.7. OSGeo QGIS

QGIS is a free and open-source cross-platform desktop GIS that supports viewing, editing, and analysis of geospatial data.

4.6. Community Open Source Projects

4.6.1. OSGeo ZOO-Project

ZOO-Project is a Web Processing Service (WPS) implementation written in C. It is an open source platform which implements the WPS 1.0.0 and WPS 2.0.0 OGC Standards.

4.6.2. TEAM Engine

The Test, Evaluation, And Measurement (TEAM) Engine is a testing facility that executes test suites developed using the TestNG framework or the OGC Compliance Test Language (CTL). It is typically used to verify specification compliance and is the official test harness of the OGC Compliance Testing Program (CITE).

4.6.3. CesiumJS

CesiumJS is an open source JavaScript library that supports visualization of 3D globes and maps on web browsers. The library supports user interaction and streaming in of 3D Tiles (an OGC Community Standard) and a variety of other formats.

4.6.4. OSGeo Leaflet

Leaflet is an open-source JavaScript library for mobile-friendly interactive maps. It works across all major desktop and mobile platforms, can be extended with a variety of plugins, and offers a well-documented API.

4.6.5. Camptocamp ogc-client

Camptocamp's ogc-client product is a Typescript library that implements several OGC Standards. The library supports the Web Map Service (WMS), Web Feature Service (WFS), and OGC API — Features standards. The library also supports the OGC API — Records candidate standard.

4.6.6. HomeAssistant-SensorThings

Home Assistant is an award-winning free/libre/open-source home automation hub software written in Python. It can interface with over a thousand kinds of IoT devices and services, with a focus on domestic environments and data privacy.

HomeAssistant-SensorThings is a client for the OGC SensorThings API Part 1: Sensing in the form of a Home Assistant plugin (or "integration" in Home Assistant jargon). It implements discovery of available sensors in an endpoint, and periodic updates of the values of those sensors.

4.6.7. Maplibre

MapLibre GL JS is an open-source library for publishing maps on websites or webview-based applications. The library uses GPU-accelerated rendering to offer client-side display of maps. The library supports client-side rendering of data, which makes it possible for an end-user to customize how a map is displayed.

4.6.8. OSGeo MDME

MDME (Model Driven Metadata Editor) is a web-based application for populating YAML or JSON oriented metadata records. The application is built using Node/Vue.js. The application

supports a variety of metadata models, for example MCF, OGC API — Records, and Data Package. The applications support extension of the internal metadata model.

4.6.9. OL-Cesium

OL-Cesium is a library that integrates OpenLayers and Cesium. The OL-Cesium library supports map contexts, raster data sources, vector data sources, map selection (selected items), and animated transitions between map and globe views.

4.6.10. OSGeo pygeometa

pygeometa provides a lightweight and Pythonic approach for users to easily create geospatial metadata in standards-based formats using simple configuration files called Metadata Control Files (MCF). The software has minimal dependencies (the installation is less than 50 kB), and provides a flexible extension mechanism leveraging the Jinja2 templating system. Leveraging the simple but powerful YAML format, pygeometa can generate metadata in numerous standards. Users can also create their own custom metadata formats which can be plugged into pygeometa for custom metadata format output. pygeometa is open source and released under an MIT license.

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

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

4.6.11. Geomapfish

GeoMapFish is web application that enables developers to build extensible web-based GIS applications. It supports functionality such as webmapping, editing, responsive templates, and customization. The software supports a variety of OGC Standards as well as the MapFish protocol for communication between client-side and server-side applications. The software is built using OpenLayers, AngularJS, ngeo, Papyrus, and other toolkits.

4.6.12. Smapshot

Smapshot is a crowdsourcing platform that enables volunteers to position historical images in 3D visualizations of geographic space, a process called ‘geolocalization’. More than a thousand volunteers have geolocalized up to 250,000 images. An OpenAPI-enabled service is available to integrate the data: https://smapshot.heig-vd.ch/development_documentation and the API and georeferencer are open source.

4.6.13. go-ogc

go-ogc is a Go library for working with OGC API services. The library includes support for OGC API — Common, OGC API — Tiles, and OGC API — Features including support for the CQL2 JSON encoding. The library is available under the Apache 2.0 license.

4.6.14. GPQ

GPQ is a utility for working with GeoParquet files. The command line interface can be used to validate a GeoParquet file, describe its metadata, and can convert data to and from GeoJSON. The utility is also available as a WebAssembly binary for use in a browser. GPQ is written in Go and is available under the Apache 2.0 license.

4.6.15. 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. It is an OGC Reference Implementation for Parts 1 and 2 of OGC API — Features.

5

RESULTS

5 RESULTS

The code sprint included multiple software libraries, OWS implementations, OGC API implementations and a variety of client applications. In addition to supporting OWS and OGC API standards, various ASF and OSGeo software products involved in the code sprint also supported a variety of OGC encoding standards. This section presents the key results from the code sprint.

5.1. Approved and Candidate OGC Standards

5.1.1. CQL2

Discussions during the code sprint led to an issue with the CQL2-JSON encoding being identified and documented. The issue is summarized below.

In the JSON encoding for CQL2, there is a need to consider whether to still use `function` instead of `op` (or `casei` / `accenti` in that particular case) for those standard functions defined by the various requirements classes of the standard making the names consistent across the various functions. That is, the standard functions are likely to be routed to actual *functions* in the implementation, just like the user-defined custom functions.

The sprint participants also noted that the `function` definition also seems overcomplicated. They suggested that instead of:

```
{
   "function": {
      "name": "lowerCase",
      "args": [ "Test" ]
   }
}
```

an option would be to use:

```
{ "function": "lowerCase", "args": [ "Test" ] }
```

as with the `op`? (or `"fn"` in the case of using a short 2 letters property like `op`).

Further detail of this issue is documented on the OGC API — Features GitHub repository. Note that the SWG meeting of 2023-05-08 considered this issue and agreed on the following resolutions:

- change the JSON encoding of the custom functions to mimic the pattern for most standardized functions using an object with the op and args members;
- change `casei` and `accenti` to the same pattern with an `op` value;

- keep `casei` and `accenti` (if `lower` or `upper` are needed, they can be added as custom (or future standardized) functions); and
- rename “Basic Spatial Operators,” “Spatial Operators,” “Temporal Operators,” and “Array Operators” to use “Functions” instead of “Operators.”

5.1.2. OGC API — Tiles

The sprint participants worked on fixing GDAL’s support of OGC API — Tiles — Part 1: Core. GDAL is an open source translator library for raster and vector geospatial data formats. The library presents a single raster abstract data model that enables the calling application to transform between a variety of data formats.

5.1.3. OGC Styles and Symbology

During this code sprint the sprint participants worked on the JSON encoding of the OGC Styles & Symbology draft candidate Standard. The work included:

- presenting the standard in detail to the GeoStyler team who then provided the editors of the standard with valuable feedback;
- a new example styling for polygonal features;
- a new JSON Schema for the encoding; and
- addition of some descriptive text for the section.

A number of GitHub Issues were created as a result of this work. This included a GitHub Issue to fix the direction of the composition arrow in Unified Modeling Language (UML) class diagrams in the draft candidate Standard. This also included an issue for updating of the naming of the enumerations of the `RelationalOperator` between the table and the UML class diagram to improve consistency. Another issue created was for addition of capabilities required for completeness. Pull requests were also created for fixing editorial issues and formatting the CartoSym JSON schema.

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

In the OGC API — Records discussions an issue was raised about the fact that some GeoJSON clients can only access information from the standard GeoJSON feature members (“id”, “geometry”, and “properties”).

Like OGC API — Features, OGC API — Records and STAC, JSON-FG also adds additional JSON members to a GeoJSON feature. While this is a valid extension of GeoJSON, the fact that such extensions may not be accessible in widely used tools and libraries is a concern. As a result of the discussions an issue was opened in the JSON-FG GitHub repository after a discussion in the JSON-FG Standards Working Group.

5.1.5. OGC GeoXACML

5.1.5.1. Purpose

Secure Dimensions implemented the GeoXACML 3.0 draft specification based on the Authzforce CE project which is an Open Source Community Edition project by FIWARE. The implementation is based on Java 11 and will be made available as the Open Source reference implementation for GeoXACML 3.0.

The implementation of a Policy Enforcement Point (PEP) during the code sprint was important for testing the GeoXACML 3.0 GeoPDP reference implementation by Secure Dimensions regarding request and response processing.

5.1.5.2. Solution

With respect to implementation, a GeoServer deployment including the default data was used as the API to be protected. From the GeoServer architecture, different possibilities exist for PEP placement. For the code sprint the solution to deploy a Apache Tomcat Filter was favored to ensure that the GeoServer installation could almost remain unchanged. The configuration of the “web.xml” could be adopted to include the “WFS Filter for GeoXACML 3.0”. Also, the filter’s JAR (Java Archive) file and dependencies must be added to “geoserver.war” (the web application archive that is deployed inside Apache Tomcat).

The implementation of the WFS Filter itself used the Authzforce XACML-SDK for Java that needs to be extended to allow the inclusion of Geometry attributes. The actual access condition to exclude all features of the Central Park was modelled in a GeoXACML 3.0 Policy. This was made possible by constructing an Obligation on Permit that returns the OGC Filter condition that the PEP needs to apply to the intercepted request.

5.1.5.3. Achievement

During the Code Sprint a demonstration was implemented that follows the XACML flow diagram shown in Figure 2.

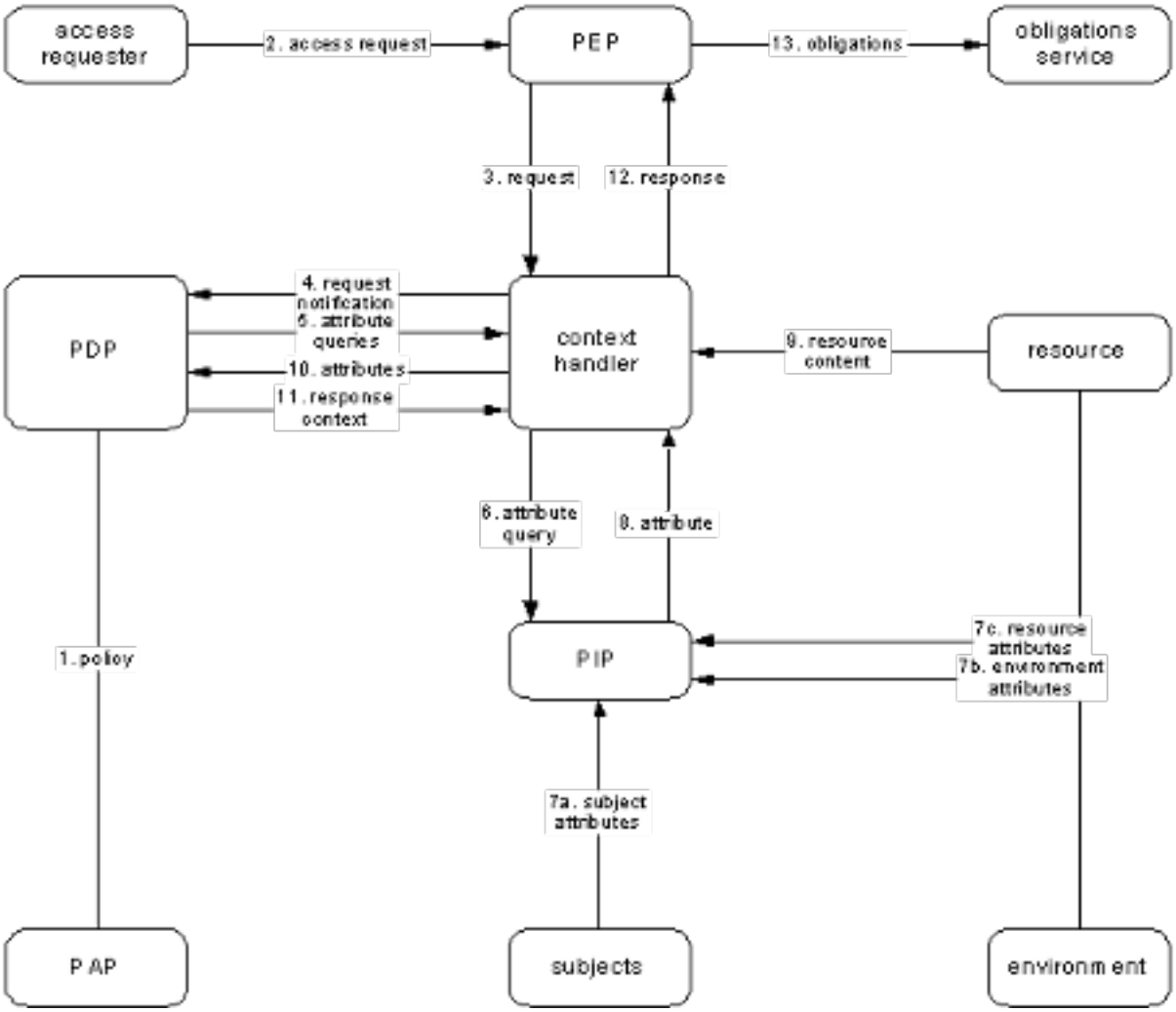

Figure 2 — XACML Flow Diagram

A GeoServer 2.23 WAR-file deployment was used create the WFS 2.0 service endpoint.

The injection of the PEP was done by updating the “geoserver.war” file manually:

- zip -u geserver.war WEB-INF/web.xml
- zip -u geoserver.war WEB-INF/lib/GeoXACMLFilter.jar

The web.xml file contains the deployment for the WFS Filter for GeoXACML 3.0.:

```
<filter>
<filter-name>GeoXACMLFilter</filter-name>
<filter-class>de.securedimensions.geoxacml3.ows.GeoServerFilter</filter-class>
<init-param>
<param-name>pdpURL</param-name>
<param-value>https://ogc.demo.secure-dimensions.de/authzforce-ce</param-value>
</init-param>
<init-param>
<param-name>pdpDomain</param-name>
<param-value>A0bdIbmGEeWhFwcKrC9gSQ</param-value>
```

```
</init-param>
</filter>
<filter-mapping>
<filter-name>Set Character Encoding</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
<!-- Uncomment following filter to enable CORS
<filter-mapping>
<filter-name>cross-origin</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
-->
<filter-mapping>
<filter-name>GeoXACMLFilter</filter-name>
<url-pattern>/wfs</url-pattern>
</filter-mapping>
```

Geoserver web.xml initializing the WFS Filter for GeoXACML 3.0 on path /wfs

During the code sprint one bug concerning the processing of XML attributes was fixed.

5.1.5.4. Demonstration

The focus during the code sprint was to demonstrate the ability to rewrite WFS requests to reflect geospatial access control decisions. In order to achieve that, a simple GeoXACML 3.0 policy was crafted using ALFA for Visual Studio Code (ALFA plugin):

```
namespace ogc_codesprint {
    import GeoXACML.*
    import Attributes.*

    obligation Filter = "urn:secd:wfs:filter"

    policyset Manhattan = "root" {
        target clause service == "WFS"
        apply permitOverrides
        policy POLY_LANDMARKS = "poly_landmarks" {
            target clause feature_type == "tiger:poly_landmarks"
            apply firstApplicable
            rule CENTRAL_PARK {
                target clause request == "GetFeature"
                permit
                condition(geometryOneAndOnly(bbox) >< "POLYGON((40.767774 -
73.981464, 40.768268 -73.981396, 40.768483 -73.981634, 40.769272 -73.981131,
40.76983 -73.980652, 40.770488 -73.980215, 40.771096 -73.9798, 40.771753 -
73.979298, 40.77241 -73.978862, 40.773018 -73.978447, 40.773708 -73.977923,
40.7743 -73.977465, 40.774859 -73.977072, 40.775565 -73.97657, 40.776288 -
73.97609, 40.776947 -73.975588, 40.777554 -73.97513, 40.778244 -73.974671,
40.778852 -73.974234, 40.779427 -73.973776, 40.782105 -73.971899, 40.782795
-73.971375, 40.78342 -73.970917, 40.783995 -73.970437, 40.784685 -73.969956,
40.785293 -73.969498, 40.785983 -73.969018, 40.786624 -73.968537, 40.787264
-73.968036, 40.787922 -73.96762, 40.78848 -73.967119, 40.789105 -73.966704,
40.789713 -73.966245, 40.790353 -73.965787, 40.79106 -73.965241, 40.791684 -
73.964804, 40.792358 -73.964281, 40.794247 -73.962905, 40.794905 -73.962403,
40.795545 -73.961966, 40.796153 -73.961529, 40.796761 -73.961049, 40.797418 -
73.960612, 40.798125 -73.960109, 40.798782 -73.959607, 40.799374 -73.959149,
40.800047 -73.95869, 40.800425 -73.958428, 40.800507 -73.958124, 40.800588
-73.957885, 40.799509 -73.955312, 40.798298 -73.95248, 40.797003 -73.94954,
40.79669 -73.94952, 40.796329 -73.949761, 40.795705 -73.950241, 40.795031 -
73.950744, 40.794374 -73.951159, 40.793684 -73.95177, 40.79306 -73.952142,
```

```
40.792419 -73.9526, 40.791729 -73.953103, 40.791154 -73.953496, 40.790414 -
73.954042, 40.789199 -73.954937, 40.788624 -73.955352, 40.78795 -73.955854,
40.78726 -73.956335, 40.786669 -73.956815, 40.786028 -73.957339, 40.78542 -
73.957732, 40.784796 -73.958212, 40.784188 -73.958627, 40.783514 -73.959086,
40.782873 -73.959588, 40.782233 -73.96009, 40.781625 -73.960548, 40.780852 -
73.961029, 40.780294 -73.961466, 40.779587 -73.961946, 40.779012 -73.962383,
40.778388 -73.962863, 40.777747 -73.963343, 40.777106 -73.963845, 40.776334
-73.964391, 40.775726 -73.964871, 40.77502 -73.965438, 40.774494 -73.965939,
40.773771 -73.966398, 40.773196 -73.966856, 40.772523 -73.967315, 40.7718 -
73.967817, 40.771225 -73.968253, 40.770585 -73.96869, 40.769992 -73.969148,
40.769368 -73.969607, 40.76871 -73.970065, 40.768135 -73.970501, 40.767511 -
73.970981, 40.766837 -73.971397, 40.766213 -73.971898, 40.765605 -73.972313,
40.764981 -73.972793, 40.764389 -73.973251, 40.764621 -73.973791, 40.765651 -
73.976428, 40.766812 -73.97926, 40.767575 -73.981008, 40.767774 -73.981464))":
geometry)
                on permit {
                    obligation Filter {
                        Attributes.Filter.operation = "Disjoint"
                        Attributes.Filter.geometry = "POLYGON((40.767774 -
73.981464, 40.768268 -73.981396, 40.768483 -73.981634, 40.769272 -73.981131,
40.76983 -73.980652, 40.770488 -73.980215, 40.771096 -73.9798, 40.771753 -
73.979298, 40.77241 -73.978862, 40.773018 -73.978447, 40.773708 -73.977923,
40.7743 -73.977465, 40.774859 -73.977072, 40.775565 -73.97657, 40.776288 -
73.97609, 40.776947 -73.975588, 40.777554 -73.97513, 40.778244 -73.974671,
40.778852 -73.974234, 40.779427 -73.973776, 40.782105 -73.971899, 40.782795
-73.971375, 40.78342 -73.970917, 40.783995 -73.970437, 40.784685 -73.969956,
40.785293 -73.969498, 40.785983 -73.969018, 40.786624 -73.968537, 40.787264
-73.968036, 40.787922 -73.96762, 40.78848 -73.967119, 40.789105 -73.966704,
40.789713 -73.966245, 40.790353 -73.965787, 40.79106 -73.965241, 40.791684 -
73.964804, 40.792358 -73.964281, 40.794247 -73.962905, 40.794905 -73.962403,
40.795545 -73.961966, 40.796153 -73.961529, 40.796761 -73.961049, 40.797418 -
73.960612, 40.798125 -73.960109, 40.798782 -73.959607, 40.799374 -73.959149,
40.800047 -73.95869, 40.800425 -73.958428, 40.800507 -73.958124, 40.800588
-73.957885, 40.799509 -73.955312, 40.798298 -73.95248, 40.797003 -73.94954,
40.79669 -73.94952, 40.796329 -73.949761, 40.795705 -73.950241, 40.795031 -
73.950744, 40.794374 -73.951159, 40.793684 -73.95177, 40.79306 -73.952142,
40.792419 -73.9526, 40.791729 -73.953103, 40.791154 -73.953496, 40.790414 -
73.954042, 40.789199 -73.954937, 40.788624 -73.955352, 40.78795 -73.955854,
40.78726 -73.956335, 40.786669 -73.956815, 40.786028 -73.957339, 40.78542 -
73.957732, 40.784796 -73.958212, 40.784188 -73.958627, 40.783514 -73.959086,
40.782873 -73.959588, 40.782233 -73.96009, 40.781625 -73.960548, 40.780852 -
73.961029, 40.780294 -73.961466, 40.779587 -73.961946, 40.779012 -73.962383,
40.778388 -73.962863, 40.777747 -73.963343, 40.777106 -73.963845, 40.776334
-73.964391, 40.775726 -73.964871, 40.77502 -73.965438, 40.774494 -73.965939,
40.773771 -73.966398, 40.773196 -73.966856, 40.772523 -73.967315, 40.7718 -
73.967817, 40.771225 -73.968253, 40.770585 -73.96869, 40.769992 -73.969148,
40.769368 -73.969607, 40.76871 -73.970065, 40.768135 -73.970501, 40.767511 -
73.970981, 40.766837 -73.971397, 40.766213 -73.971898, 40.765605 -73.972313,
40.764981 -73.972793, 40.764389 -73.973251, 40.764621 -73.973791, 40.765651 -
73.976428, 40.766812 -73.97926, 40.767575 -73.981008, 40.767774 -73.981464))":
geometry
                    }
                }
            }
            rule ALL_PERMIT {
                permit
            }
        }

    }
```

}

GeoXACML 3.0 PolicySet described in ALFA

The GeoXACML 3.0 policy is structured in a simple way:

- PolicySet (Manhattan) matches “service == WFS”
- Policy (POLY_LANDMARKS) matches “typesNames == tiger:poly_landmarks”
- Rule (CENTRAL_PARK) matches “request == GetFeature”
- The Rule Condition contains is geospatial “BBOX Intersects Polygon(...)”

Any request that matches the condition results in the decision “Permit” with the Obligation “urn:secd:filter”. As specified in the XACML 3.0 specification, a PEP must enforce the decision including all obligations. The processing of this filter obligation provides the missing information to construct the WFS Filter (disjoint Central Park).

The result of this processing can be visualized with QGIS (WFS Layer):

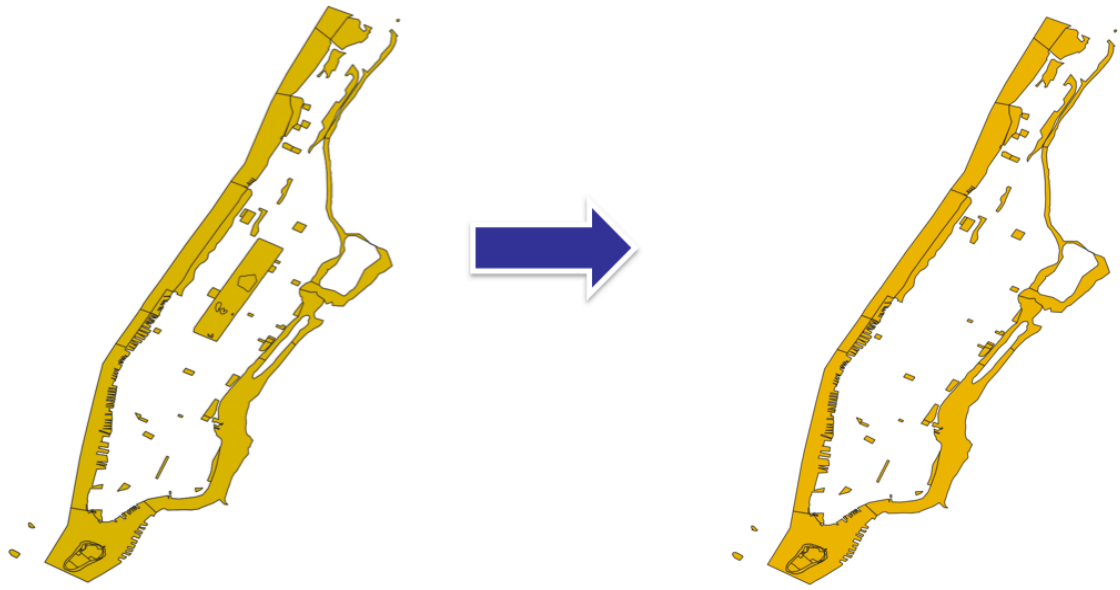

Figure 3 — Left: Feature type “poly_landmarks” without PEP → Central Park feature(s) are included; Right: Feature type “poly_landmarks” with PEP → Central Park feature(s) are excluded!

The implementation of the Filter obtains the information from the HTTP request:

```
SubjectCategory subjectCat = new SubjectCategory();
ResourceCategory resourceCat = new ResourceCategory();
ActionCategory actionCategory = new ActionCategory();
EnvironmentCategory environmentCategory = new EnvironmentCategory();

AttributeValueType serviceType = new
AttributeValueType(Arrays.asList(httpRequest.getParameter("SERVICE")),
XACMLDatatypeId.STRING.value(), null);
Attribute service = new Attribute(Arrays.asList(serviceType),"urn:ogc:ows:
service", "", false);
```

```
resourceCat.addAttribute(service);
```

Sample code for obtaining information from the HTTP request

Using the XACML-SDK for Java from Authzforce, the response from the PDP can be obtained in a few lines of code:

```
Request xacmlRequest = Utils.createXacmlRequest(Arrays.asList(subjectCat),
 Arrays.asList(resourceCat), Arrays.asList(actionCategory),
Arrays.asList(environmentCategory));

ResponsesFactory xacmlResponse = pdp.getAuthZ(subjectCat, resourceCat,
actionCategory, environmentCategory);
for (Response r : xacmlResponse.getResponses()) {
    LOGGER.info("XACML Response: " + r.toString());
    DecisionType decision = r.getDecision();
    LOGGER.info("XACML Decision: " + decision.toString());
    LOGGER.info("decision: " + decision.value());
    for (Obligation obligation : r.getObligations().getObligations()) {
        if (obligation.getObligationId().equalsIgnoreCase("urn:secd:wfs:
filter")) {
            for (AttributeAssignment aa :
obligation.getAttributeAssignments()) {
                if (aa.getAttributeId().equalsIgnoreCase("urn:secd:filter:
geometry")) {
                    filterGeometry = aa.getContent().get(0).toString();
                }
                if (aa.getAttributeId().equalsIgnoreCase("urn:secd:filter:
operation")) {
                    filterOperation = aa.getContent().get(0).toString();
                }
            }

        }
    }
}
```

Sample code for obtaining the response from the PDP

5.1.5.5. Lessons Learned

The Tomcat Filter implementation was based on Java 11. The existing XACML-SDK for Java was available for Java 8. Due to deprecation of javax classes in Java 11, JAXB related functionality had to be updated. The use of GeoTools to create the Filter programmatically could not be achieved. The unresolved problem was that the XML encoder did not include the CRS into the GML part of the spatial filter. Examples and documentation were found to only cover non-spatial examples or the BBOX Filter.

For implementing the use case, a Disjoint filter with a GML3 geometry had to be constructed. After removing GeoTools completely, a simple string template was used:

“<fes:OPERATION><fes:ValueReference>the_geom</fes:ValueReference>GEOMETRY</
fes: OPERATION >” where the GEOMETRY was constructed from the response by the PDP (Obligation attributes urn:secd:filter:geometry and urn:secd:filter:operation).

```
<?xml version='1.0' encoding='UTF-8'?><ns4:Response xmlns:ns6="http://
authzforce.github.io/pap-dao-flat-file/xmlns/properties/3.6" xmlns:ns5="http://
authzforce.github.io/core/xmlns/pdp/8" xmlns:ns4="urn:oasis:names:tc:xacml:3.0:
```

```
core:schema:wd-17" xmlns:ns3="http://www.w3.org/2005/Atom" xmlns:ns2="http://
authzforce.github.io/rest-api-model/xmlns/authz/5"><ns4:Result><ns4:Decision>
Permit</ns4:Decision><ns4:Obligations><ns4:Obligation ObligationId="urn:secd:
wfs:filter"><ns4:AttributeAssignment AttributeId="urn:secd:filter:operation"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource" DataType=
"http://www.w3.org/2001/XMLSchema#string">Disjoint</ns4:AttributeAssignment>
<ns4:AttributeAssignment AttributeId="urn:secd:filter:geometry" Category=
"urn:oasis:names:tc:xacml:3.0:attribute-category:resource" DataType="urn:
ogc:def:geoxacml:3.0:data-type:geometry">POLYGON ((40.767774 -73.981464,
40.768268 -73.981396, 40.768483 -73.981634, 40.769272 -73.981131, 40.76983
-73.980652, 40.770488 -73.980215, 40.771096 -73.9798, 40.771753 -73.979298,
40.77241 -73.978862, 40.773018 -73.978447, 40.773708 -73.977923, 40.7743 -
73.977465, 40.774859 -73.977072, 40.775565 -73.97657, 40.776288 -73.97609,
40.776947 -73.975588, 40.777554 -73.97513, 40.778244 -73.974671, 40.778852 -
73.974234, 40.779427 -73.973776, 40.782105 -73.971899, 40.782795 -73.971375,
40.78342 -73.970917, 40.783995 -73.970437, 40.784685 -73.969956, 40.785293 -
73.969498, 40.785983 -73.969018, 40.786624 -73.968537, 40.787264 -73.968036,
40.787922 -73.96762, 40.78848 -73.967119, 40.789105 -73.966704, 40.789713 -
73.966245, 40.790353 -73.965787, 40.79106 -73.965241, 40.791684 -73.964804,
40.792358 -73.964281, 40.794247 -73.962905, 40.794905 -73.962403, 40.795545 -
73.961966, 40.796153 -73.961529, 40.796761 -73.961049, 40.797418 -73.960612,
40.798125 -73.960109, 40.798782 -73.959607, 40.799374 -73.959149, 40.800047
-73.95869, 40.800425 -73.958428, 40.800507 -73.958124, 40.800588 -73.957885,
40.799509 -73.955312, 40.798298 -73.95248, 40.797003 -73.94954, 40.79669 -
73.94952, 40.796329 -73.949761, 40.795705 -73.950241, 40.795031 -73.950744,
40.794374 -73.951159, 40.793684 -73.95177, 40.79306 -73.952142, 40.792419 -
73.9526, 40.791729 -73.953103, 40.791154 -73.953496, 40.790414 -73.954042,
40.789199 -73.954937, 40.788624 -73.955352, 40.78795 -73.955854, 40.78726 -
73.956335, 40.786669 -73.956815, 40.786028 -73.957339, 40.78542 -73.957732,
40.784796 -73.958212, 40.784188 -73.958627, 40.783514 -73.959086, 40.782873
-73.959588, 40.782233 -73.96009, 40.781625 -73.960548, 40.780852 -73.961029,
40.780294 -73.961466, 40.779587 -73.961946, 40.779012 -73.962383, 40.778388 -
73.962863, 40.777747 -73.963343, 40.777106 -73.963845, 40.776334 -73.964391,
40.775726 -73.964871, 40.77502 -73.965438, 40.774494 -73.965939, 40.773771
-73.966398, 40.773196 -73.966856, 40.772523 -73.967315, 40.7718 -73.967817,
40.771225 -73.968253, 40.770585 -73.96869, 40.769992 -73.969148, 40.769368 -
73.969607, 40.76871 -73.970065, 40.768135 -73.970501, 40.767511 -73.970981,
40.766837 -73.971397, 40.766213 -73.971898, 40.765605 -73.972313, 40.764981 -
73.972793, 40.764389 -73.973251, 40.764621 -73.973791, 40.765651 -73.976428,
40.766812 -73.97926, 40.767575 -73.981008, 40.767774 -73.981464))</ns4:
AttributeAssignment></ns4:Obligation></ns4:Obligations>
```


Disjoint filter with a geometry constraint

The rewritten Filter was then processed by GeoServer:


```
typeNames[0] = {http://www.census.gov}poly_landmarks
srsName = urn:ogc:def:crs:EPSG::4326
filter = [[ the_geom within POLYGON ((40.46203574999999 -74.35610985937501,
40.46203574999999 -74.103704953125, 40.674587249999995 -74.103704953125,
40.674587249999995 -74.35610985937501, 40.46203574999999 -74.35610985937501))
] AND [ the_geom disjoint POLYGON ((40.767774 -73.981464, 40.768268 -
73.981396, 40.768483 -73.981634, 40.769272 -73.981131, 40.76983 -73.980652,
40.770488 -73.980215, 40.771096 -73.9798, 40.771753 -73.979298, 40.77241 -
73.978862, 40.773018 -73.978447, 40.773708 -73.977923, 40.7743 -73.977465,
40.774859 -73.977072, 40.775565 -73.97657, 40.776288 -73.97609, 40.776947 -
73.975588, 40.777554 -73.97513, 40.778244 -73.974671, 40.778852 -73.974234,
40.779427 -73.973776, 40.782105 -73.971899, 40.782795 -73.971375, 40.78342 -
73.970917, 40.783995 -73.970437, 40.784685 -73.969956, 40.785293 -73.969498,
40.785983 -73.969018, 40.786624 -73.968537, 40.787264 -73.968036, 40.787922
-73.96762, 40.78848 -73.967119, 40.789105 -73.966704, 40.789713 -73.966245,
40.790353 -73.965787, 40.79106 -73.965241, 40.791684 -73.964804, 40.792358 -
73.964281, 40.794247 -73.962905, 40.794905 -73.962403, 40.795545 -73.961966,
```

```
40.796153 -73.961529, 40.796761 -73.961049, 40.797418 -73.960612, 40.798125
-73.960109, 40.798782 -73.959607, 40.799374 -73.959149, 40.800047 -73.95869,
40.800425 -73.958428, 40.800507 -73.958124, 40.800588 -73.957885, 40.799509
-73.955312, 40.798298 -73.95248, 40.797003 -73.94954, 40.79669 -73.94952,
40.796329 -73.949761, 40.795705 -73.950241, 40.795031 -73.950744, 40.794374
-73.951159, 40.793684 -73.95177, 40.79306 -73.952142, 40.792419 -73.9526,
40.791729 -73.953103, 40.791154 -73.953496, 40.790414 -73.954042, 40.789199
-73.954937, 40.788624 -73.955352, 40.78795 -73.955854, 40.78726 -73.956335,
40.786669 -73.956815, 40.786028 -73.957339, 40.78542 -73.957732, 40.784796 -
73.958212, 40.784188 -73.958627, 40.783514 -73.959086, 40.782873 -73.959588,
40.782233 -73.96009, 40.781625 -73.960548, 40.780852 -73.961029, 40.780294 -
73.961466, 40.779587 -73.961946, 40.779012 -73.962383, 40.778388 -73.962863,
40.777747 -73.963343, 40.777106 -73.963845, 40.776334 -73.964391, 40.775726
-73.964871, 40.77502 -73.965438, 40.774494 -73.965939, 40.773771 -73.966398,
40.773196 -73.966856, 40.772523 -73.967315, 40.7718 -73.967817, 40.771225 -
73.968253, 40.770585 -73.96869, 40.769992 -73.969148, 40.769368 -73.969607,
40.76871 -73.970065, 40.768135 -73.970501, 40.767511 -73.970981, 40.766837 -
73.971397, 40.766213 -73.971898, 40.765605 -73.972313, 40.764981 -73.972793,
40.764389 -73.973251, 40.764621 -73.973791, 40.765651 -73.976428, 40.766812 -
73.97926, 40.767575 -73.981008, 40.767774 -73.981464)) ]]
```

Rewritten Filter

5.1.5.6. Follow-Up

The OGC API — Features endpoint was not used as it is not yet possible to pass a Filter parameter. Once the Filter capability is standardized in OGC API — Features — Part 3, the implementation for this demonstration could be adopted accordingly. The implementation would need to be adopted to obtain relevant information such as the feature-type, the request BBOX, etc., according to OGC API — Features.

5.1.6. OGC GeoAPI

The work done during this code sprint was not on GeoAPI itself, but on a usage of it. GeoAPI is the interface between the Java runner of GIGS tests (Geospatial Integrity of Geoscience Software) and PROJ-JNI. Both projects have been updated for enabling the execution of a greater range of GIGS tests on PROJ. The changes consisted in upgrading the way that client codes discover GeoAPI implementations. That discovery is done through the Java *Service Providers* mechanism. However, that mechanism changed significantly between Java 8 and Java 9. The new mechanism in Java 9, in addition to being more secure, also provides more flexibility in the way to discover services. This flexibility has been exploited during this code sprint for filling some holes in the GIGS test coverage.

The Apache SIS project, which is another GeoAPI implementation, has not yet been updated to Java 9+ modularization. That update will take more time because SIS is made of many modules, while PROJ-JNI is a single module. But this testbed experimented how the work can been done. An outcome of this testbed is that a small change will be needed in GeoAPI for working with the more security-constrained way that Java 9+ discovers services and resources.

5.1.7. OGC API — Records

Edits to the candidate standard in preparation for the code sprint included:

- update of core/openapi/schemas/language.yaml;
- setting of the default language direction to ltr;
- Removal of obsolete files; and
- Removal of all requirements/recommendations copied from OGC API — Features — Part 1.

A sample of the edits made to the candidate standard during the code sprint is as follows.

- Addition of a providers' array to align with STAC.
- Other changes to various schemas and recommendations towards closer STAC harmonization.
- Addition of a time member to the Record example and updates to the Contact information.
- Addition of requirements for the logo member.
- Change of the deliveryPoint member to an array of string to allow multiple address.
- Updates to the theme model.
- Fixing of the nullability of the time property and add resolution.

The OGC API — Records team held two breakouts during the code sprint focusing on advancing the specification for review by the OGC Architecture Board (OAB). The discussions during the breakout sessions included:

- Extension for faceted searching
- Record model: contacts and roles
- STAC harmonization.

5.1.8. OGC API — Features

Before the code sprint a new proposal for handling feature schemas had been prepared for discussion in the OGC Features API Standards Working Group and at the code sprint.

In current drafts, a JSON schema for the GeoJSON encoding of the features is returned by the API as the feature schema. That approach has several disadvantages as follows.

- Not all APIs can support GeoJSON (e.g., for 3D buildings).
- There was a need to have different schemas for different operations. For example, the server may return features with a (slightly) different schema than it expects when a new feature is created or updated.
- Validation of content is one use case for a schema, but for other use cases an encoding-agnostic logical schema is more beneficial and easier to use. For the queryables and sortables resources such a schema is already used.
- Another example is that of feature relationships that should be expressed as such in the schema but that can be encoded in various ways in the data depending on the format or the use case.

The proposal addresses these issues by:

- providing an encoding-agnostic logical schema of the features in a collection (expressed in JSON Schema, using the same requirements and recommendations as in Part 3 for the Queryables resource);
- adding an additional keyword to identify special roles that a property may have (e.g., identify the feature id, the primary geometry, etc);
- using the JSON Schema keywords `readOnly` and `writeOnly` to identify properties that are only relevant for GET (read-only) or POST/PUT/PATCH (write-only) requests;
- adding support for properties that are a reference to another feature in the same API or another API; and
- specifying a new query parameter `profile` to support requesting different representations of such feature references in the data.

The proposal was discussed by sprint participants in a breakout session. The GitHub issue was updated with the results of the discussion. There was agreement to start work on a pull request for OGC API — Features, to test the approach in implementations, and to present the approach to other OGC API working groups at the June 2023 OGC Member Meeting.

Participants from interactive instruments started to work on an implementation in ldproxy during the code sprint. The implementation was completed in the weeks after the code sprint.

The approach was presented and discussed in the OGC API track during the June 2023 OGC Member Meeting. The approach was supported by other OGC API working groups. As a result the Standards Working Group decided to use the approach as the starting point for a new part (Part 5: Schemas).

Other OGC API — Features topics that were discussed with participants from open source projects included the following.

- Discussion with Smapshot developers about the relationship between GeoPose and features and how to provide GeoPose via building blocks from OGC API — Features.

- Support for faceted search, i.e., return not only a page of selected features, but also statistical information about important values of selected properties and the number of features that have that value. This information can be used to interactively "drill down" to find features of interest. A proposal currently exists in OGC API — Records.
- For clients that interact with a Features API, it would be beneficial if APIs would provide not only a textual GeoJSON encoding, but also a more efficient binary encoding. Currently no conformance class exists for such a feature encoding in OGC API — Features. This is a topic that should be discussed in the Standards Working Group.

5.1.9. OGC API — Environmental Data Retrieval

In the OGC MetOcean Domain Working Group (DWG) there is interest in establishing an OGC API approach for Publish/Subscribe (Pub/Sub) operations. This is because alerts have to be sent out instantly when severe weather events occur. Members of the MetOcean DWG have been developing a Discussion Paper on Pub/Sub operations which was presented at the OGC Member Meeting in Frascati, Italy in February 2023. It is envisaged that the Discussion Paper will lead to the specification of a conformance class in the OGC API — Environmental Data Retrieval (EDR) standard that can also be used by other OGC API Standards as well.

The Discussion Paper describes a number approaches of how to support event-driven functionality in OGC API Standards. The Discussion Paper specifies a prototype API. The next step is turning that prototype into a conformance class. The basic premises behind the specification is to be generic. The specification recommends use of AsyncAPI to advertize an event-driven capability. The specification also recommends use of a well-known encoding, for example, JSON. Finally, the specification recommends use of the endpoint of the building blocks used for OGC API when advertizing the event-driven capability.

An initial attempt at generic conformance classes is being developed in a draft pull request to OGC API — EDR.

The generic nature of the draft at the time of the code sprint is a challenge for subscribers as the content of the messages is undefined. As a result, different publishers will design messages in different ways and provider-specific subscription clients have to be developed. This resulted in a proposal for adding a new conformance class for a generic message payload, which was added to the OGC API — EDR pull request after the code sprint.

During the code sprint pygeoapi was updated with an implementation of the draft Pub/Sub extension.

Work on a new ldproxy Pub/Sub module for feature resources was started based on the draft Pub/Sub extension. Two open-source Java libraries for MQTT were analyzed (Eclipse Paho and HiveMQ). HiveMQ was selected for the development, mainly because of its support for reactive programming. Only the initial framework was set up during the code sprint. The work will be continued in OGC Testbed-19.

5.2. Other Specifications

5.2.1. AsyncAPI

The maintainers of pygeoapi worked on support for AsyncAPI. A discussion of the pygeoapi results is presented in Clause 6.5. The landing page of the pygeoapi instance was modified to enable it to advertise links to both the OpenAPI definition document and the AsyncAPI definition document. Figure 4 shows a screenshot of the landing page of a pygeoapi instance that advertises both an OpenAPI and an AsyncAPI definition document.

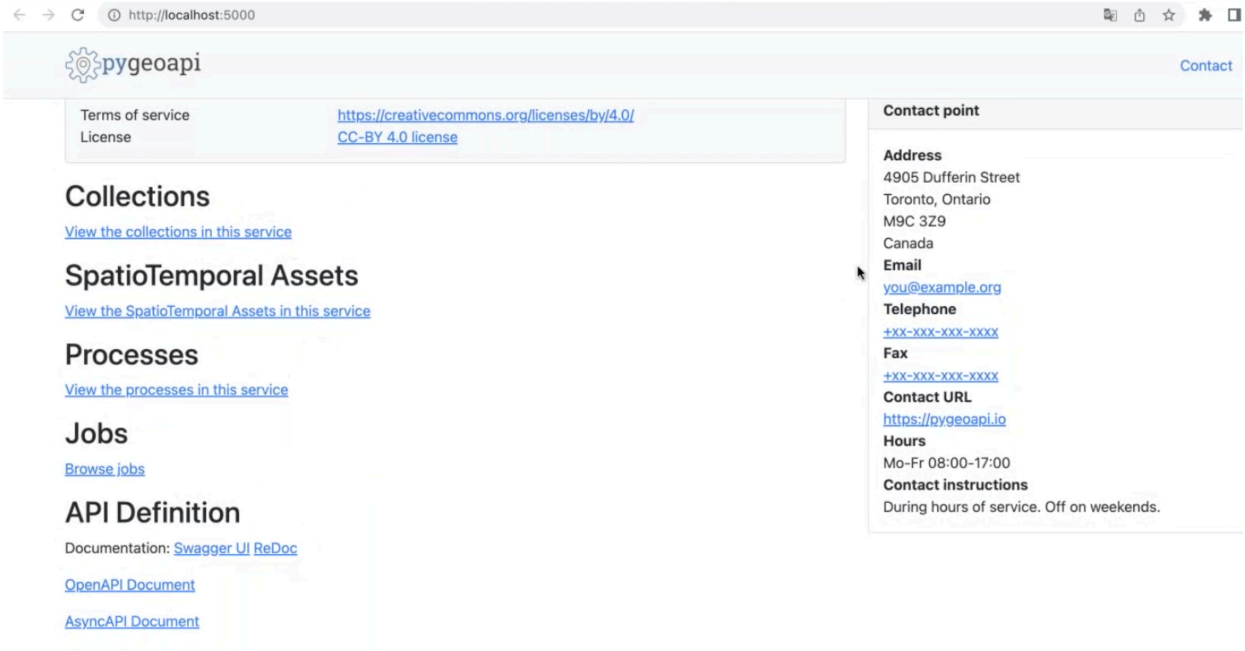

Figure 4 — Screenshot of a pygeoapi landing page

An extract of the JSON-encoding of the landing page is shown in the following listing. The advertisement of the OpenAPI and AsyncAPI definition documents in the same landing page highlighted the need to have a way to distinguish between the two types of links. OGC API Standards use `rel` and `type` link-params (link parameters) to describe the type of resource that is referenced by a link. Therefore, there is a need for a media type that uniquely identifies an AsyncAPI document that is encoded in JSON. This issue was posted on the OGC API — Common issues board and also on the AsyncAPI issues board. Also related to this issue is the need to be able to uniquely distinguish between an HTML-encoded API definition document that is based on OpenAPI and one that is based on AsyncAPI. This second issue was posted on the OGC API — Common repository.

```
{
    "links": [
        {
            "rel": "self",
            "type": "application/json",
            "title": "This document as JSON",
```

```
            "href": "http://localhost:5000?f=json"
        },
        {
            "rel": "alternate",
            "type": "application/ld+json",
            "title": "This document as RDF (JSON-LD)",
            "href": "http://localhost:5000?f=jsonld"
        },
        {
            "rel": "alternate",
            "type": "text/html",
            "title": "This document as HTML",
            "href": "http://localhost:5000?f=html",
            "hreflang": "en-US"
        },
        {
            "rel": "service-desc",
            "type": "application/vnd.oai.openapi+json;version=3.0",
            "title": "The OpenAPI definition as JSON",
            "href": "http://localhost:5000/openapi"
        },
        {
            "rel": "service-desc",
            "type": "application/json",
            "title": "AsyncAPI document in JSON",
            "href": "http://localhost:5000/asyncapi"
        },
        {
            "rel": "service-doc",
            "type": "text/html",
            "title": "The OpenAPI definition as HTML",
            "href": "http://localhost:5000/openapi?f=html",
            "hreflang": "en-US"
        },
        {
            "rel": "conformance",
            "type": "application/json",
            "title": "Conformance",
            "href": "http://localhost:5000/conformance"
        },
        {
            "rel": "data",
            "type": "application/json",
            "title": "Collections",
            "href": "http://localhost:5000/collections"
        },
        {
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/processes",
            "type": "application/json",
            "title": "Processes",
            "href": "http://localhost:5000/processes"
        },
        {
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/job-list",
            "type": "application/json",
            "title": "Jobs",
            "href": "http://localhost:5000/jobs"
        }
    ],
    "title": "pygeoapi Demo instance - running latest GitHub version",
    "description": "pygeoapi provides an API to geospatial data"
}
```

5.3. ASF Apache Projects

5.3.1. Apache Baremaps

Multiple issues were identified and documented in the Apache Baremaps repository. For example, one issue identified the need to implement support for OGC API — Features. Another issue identified possible use of an unknown WKB(Well-Known Binary) type. The issue occurs when using a protobuf encoded version of the Berlin data from OpenStreetMap. Another issue, also discovered through use of the Berlin data, is that some of the polygons did not form closed rings.

5.3.2. Apache Sedona

Apache Sedona leverages a number of open-source libraries, including GeoTools, to execute some spatial operations. However, as an Apache project, Sedona is striving to reduce its reliance on GeoTools due to licensing issues. In order to address this challenge, Sedona is migrating affected functions to Apache SIS.

In this code sprint, Sedona maintainers concentrated on replacing GeoTools' usage in two key functions within Sedona: ST_Transform and the GeoTiff reader. Their focus was on demonstrating how to migrate these functions to Apache SIS, providing a comprehensive tutorial on the necessary steps involved.

5.3.3. Apache SIS

Maintainers of Apache SIS presented a tutorial in the sprint's Mentor Stream on how to integrate GeoAPI, PROJ-JNI, and Apache SIS into a Maven project showing how to set up the project in such a way that coordinate operations can be performed without exposing the implementation (PROJ-JNI in this example) and how a project can offer developers a separation between public API (GeoAPI) and implementation details (PROJ-JNI).

The participants also demonstrated how to replace a PROJ-JNI implementation by Apache SIS. The tutorial included a step-by-step explanation of coordinate operations using a scenario described in OGC Testbed 18: Reference Frame Transformation Engineering Report. The engineering report describes how OGC Standards can be used for describing a transformation from coordinates relative to a spacecraft (e.g., Voyager 2) to coordinates relative to Earth.

The tutorial also demonstrated advanced referencing services that are well beyond classical map projections, how OGC Standards support them, and how Apache SIS implements them. Finally, the tutorial demonstrated basic raster operations using data conformant to the OGC GeoTIFF and netCDF Standards. A multi-dimensional data cube was built and scalability with large files was shown. It is envisaged that demonstration of these capabilities will lead to discussion about what the coverage package of the OGC GeoAPI Standard may look like.

5.4. OSGeo Projects

5.4.1. GeoNetwork

5.4.1.1. Support for OGC API Records in GeoNetwork-UI

Participants from Camptocamp worked on the following tasks related to the User Interface (UI) of GeoNetwork:

- creating an abstraction level between the search-related components of GeoNetwork-UI and the actual search backend used for metadata records; and
- implementing a search backend compatible with OGC API — Records.

Eventually, a GeoNetwork-UI application such as the Datahub could be able to provide a search interface over any implementation of OGC API — Records. Some features currently provided by GeoNetwork 4.x might not be available anymore, but basic search functionalities should still offer a valuable user experience.

Pull request: https://github.com/geonetwork/geonetwork-ui/pull/464

5.4.1.2. Filtering improvements to GeoNetwork-UI

Participants from Astun Technology worked on the following task related to the GeoNetwork-UI application Datahub:

- expanding the available search filters for Datahub to include metadata standards and INSPIRE keywords.

Pull request: https://github.com/geonetwork/geonetwork-ui/pull/477

5.4.2. OSGeo GeoServer

The GeoServer team focused on separating the implementation into individually downloadable modules. This work is required in anticipation of CQL2 being finalized and enabling the first public release of a GeoServer ogcapi module that supports OGC API Standards.

Some of the outputs included:

- a pull reqeust splitting ogcapi module implementation into individual downloads;

- a pull request of a nightly build docker image allowing public testing of ogcapi modules above; and
- a pull request providing fixes to the OGC API — Tiles implementation, based on the draft CITE tests for it.

The GeoServer maintainers also checked in with the status of CQL2 which remains under active development and are waiting on this activity before publishing the ogcapi-features module (supporting OGC API — Features) for the general public.

5.4.3. OSGeo OpenLayers

OpenLayers contributors discussed topics related to vector feature styling and rendering at the code sprint. The discussions were to support on-going development of a new WebGL vector renderer, as well as work on a new internal feature representation and style encoding syntax to be supported by the new renderer.

OpenLayers currently allows for very flexible feature style generation by allowing users to provide an arbitrary function to generate symbolizers at render time. This limits the efficiency of vector rendering as the library is currently unable to evaluate these style functions outside the main thread. OpenLayers contributors made progress at the code sprint in designing a new encoding for rule-based styling that allows flexibility in providing expressions to generate symbolizer values while still allowing for efficient evaluation of these expressions in a worker or on the Graphics Processing Unit(GPU) of a computer.

OpenLayers contributors will likely plan another dedicated sprint to continue this work and implement the new styling design in the WebGL vector renderer in addition to the existing Canvas (2D) renderer.

5.4.4. OWSLib

The OWSLib project updated support for OGC API — Features — Part 4: Create, Replace, Update, and Delete, and upgraded OWSLib's public documentation to use Read the Docs as well as continuous deployment workflows for automated documentation updates.

5.4.5. OSGeo pycsw

The pycsw project upgraded support for STAC API to v1.0.0, and added updates/fixes to its support for OGC API — Features — Part 3: Filtering and Common Query Language (CQL2).

5.4.6. OSGeo pygeoapi

The pygeoapi project completed the implementation of a feature provider for the ERDDAP data server. ERDDAP is a data server that provides a simple, consistent way to download subsets of scientific datasets in common file formats, graphs, and maps. This new pygeoaopi feature

was created by WMO as part of the https://docs.wis2box.wis.wmo.int project and has been promoted to be included in the pygeoapi core.

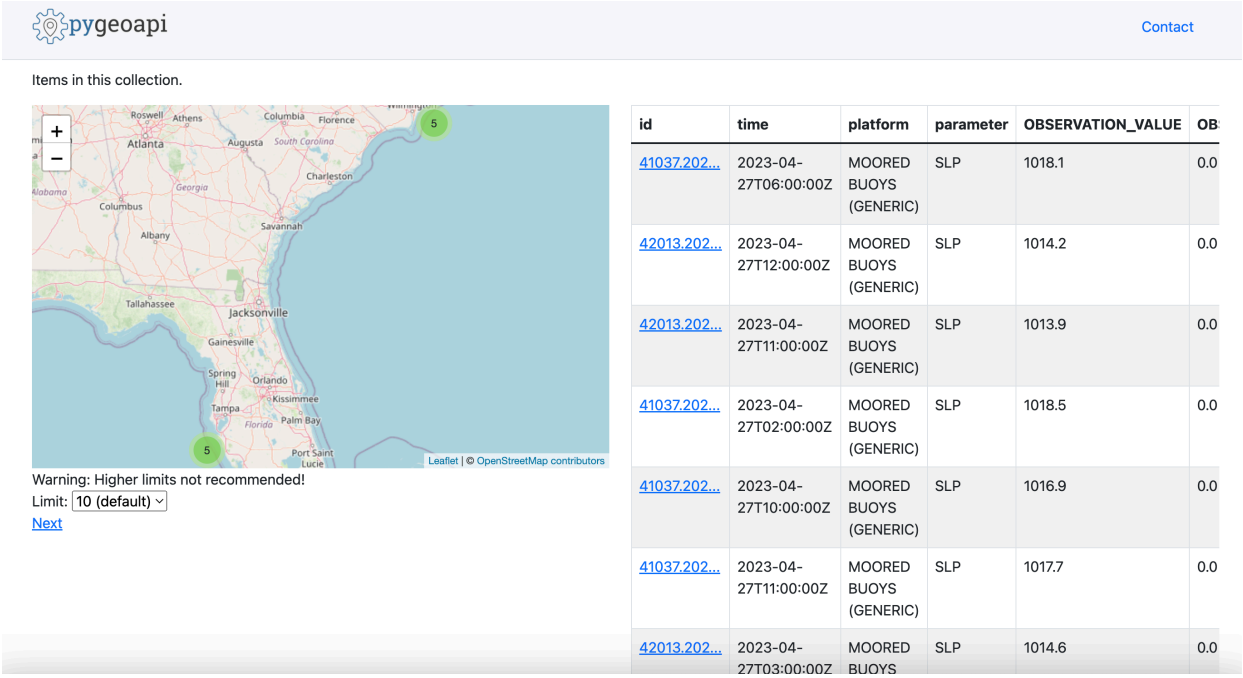

Figure 5 — Screenshot of the pygeoapi OGC API - Features provider for the ERDDAP data server.

An update was also made in pygeoapi to highlight licensing when available as part of a collection definition's link object (where `rel=license`).

The pygeoapi project implemented the draft OGC API — Pub/Sub conformance class of OGC API — Environmental Data Retrieval standard (see also the original discussion paper (23-013)).

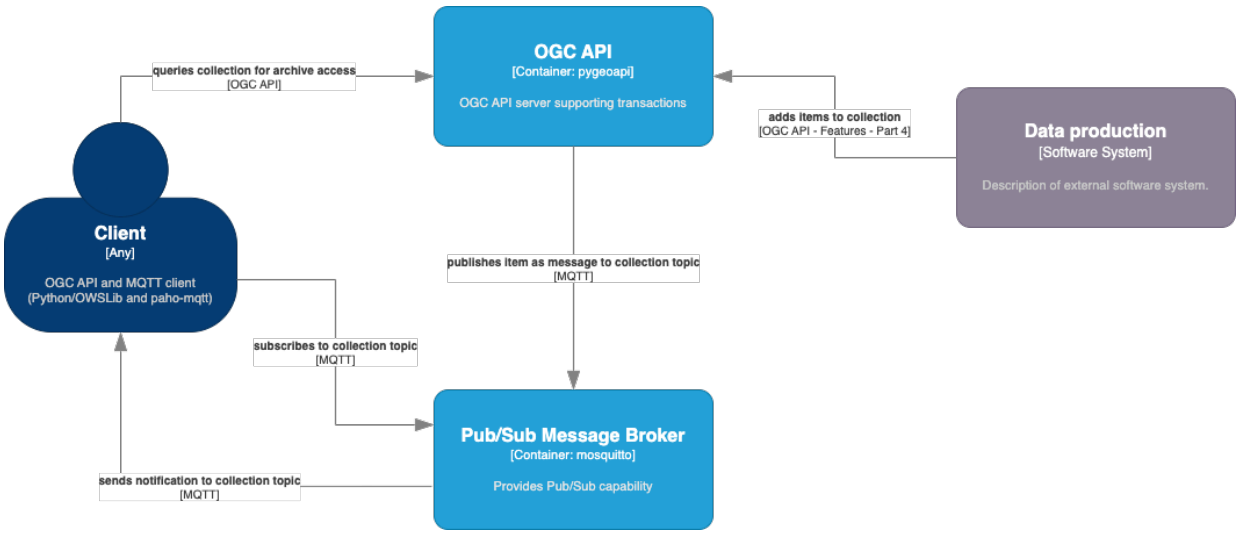

Figure 6 — pygeoapi OGC API - Pub/Sub architecture.

Using OWSLib support for OGC API — Features — Part 4: Create, Replace, Update, and Delete, the workflow entailed publishing new features via transactions which then triggered push

notifications via the MQTT protocol. Pub/Sub capabilities were made available via the AsyncAPI specification leveraging the OGC API endpoints as subscription channels.

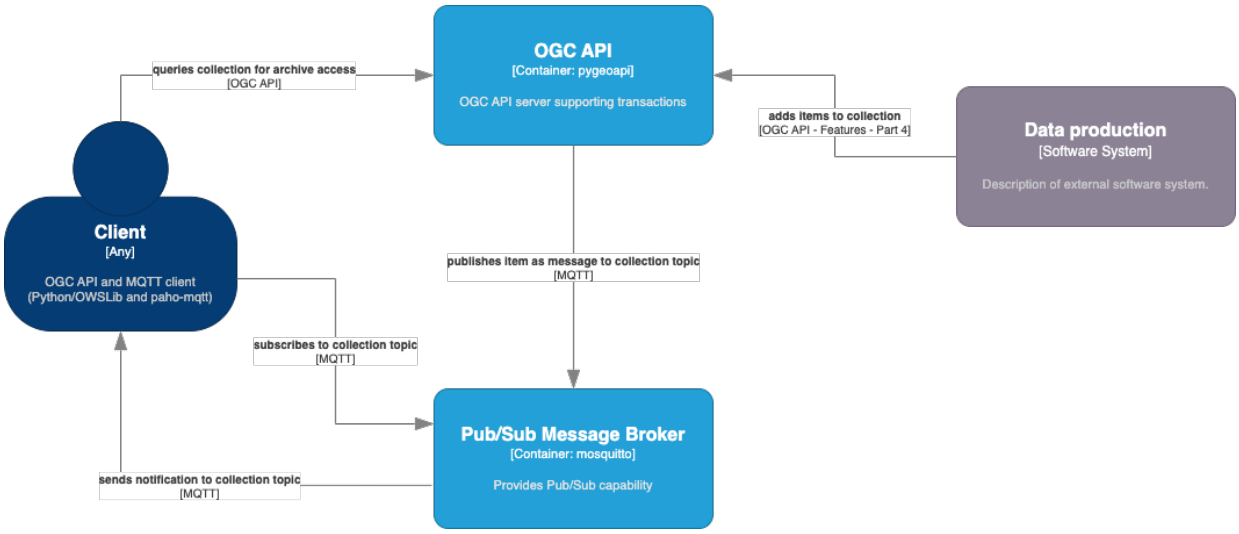

Figure 7 — pygeoapi OGC API - Pub/Sub architecture.

5.4.7. OSGeo QGIS

The sprint participants identified and documented a number of issues on the QGIS GitHub repository. An issue was documented recommending implementation of support of the record format of the OGC API — Records candidate Standard in QGIS. Such support would enable users to read and write layer metadata using this format. One option could be to add a translator class from the current QgsProjectMetadata class, which would be called when serializing/deserializing information. Another option could be to replace the current QGIS metadata internal format with OGC API — Records format.

The QGIS metadata internal format is based on Dublin Core. It was developed with the goal of creating something simple (standards based), that could be extended in the future. At that time, OGC API — Records did not exist. OGC API — Records defines a record building block which shares the same goals as the QGIS metadata schema: being very simple and extensive. Unlike Dublin Core, which is a generic metadata schema, the OGC API — Records schema was designed for geospatial data and is envisaged to be at the core of the modern generation geospatial catalogs.

Another issue that the sprint participants worked on related to supporting OGC API — Tiles in QGIS. The idea was to allow users to use the browser panel or the layer menu to connect to a conformant server and to pull collections from an OGC API — Tiles implementation. Prior to the code sprint, QGIS already supported some aspects of this use case. So the code sprint provided an opportunity for bug fixes and enhancements. Some of the fixes were implemented on GDAL, which QGIS leverages for accessing OGC API — Tiles implementations.

5.5. Community Open Source Projects

5.5.1. OSGeo ZOO-Project

The ZOO-Project worked on moving forward OSGeo Incubation as well as the remaining tasks as part of the Project Graduation Checklist to become an OSGeo project. Discussion with other participants regarding OGC API — Processes — Part 3: Workflows and Chaining led to some initial work on ZOO Project's support for this candidate standard.

5.5.2. TEAM Engine

The aim of the code sprint was to migrate TEAM Engine to Java 17 and to get it running with the executable test suite of OGC API — Features. This is an important part of the upcoming TEAM Engine 6.0 release which will focus on updating Java to version 17, Tomcat to version 10, and Maven dependencies including Java libraries and Maven plugins.

First, a test setup was built using a Dockerfile which contains Java 17, Tomcat 7, the latest build of TEAM Engine 6.0-SNAPSHOT, and the OGC API — Features executable tests suite compiled with Java 8.

Then, the code of TEAM Engine was migrated to Java 17 with the aim that the complete build is successful. This work was completed during the code sprint and it was possible to build and compile TEAM Engine with command `mvn clean install site -DskipTests` using Java 17. All results were documented in issue 511.

Afterward, a TEAM Engine instance compiled with Java 17 was tested with the OGC API — Features executable test suite using the above described test setup. No errors were detected during testing. The TEAM Engine webapp started, a login via Web Browser Interface was possible, and a test run with OGC API — Features executable test suite could be executed. The generated report was displayed as expected and test details could be viewed. Thus, the aim of the code sprint was reached.

In the remaining time, the work concentrated on further tasks that are important for the TEAM Engine 6.0 release.

Identified open tasks were documented in separate issues:

- Unit test ImageParserTest.parsePNG_noAlphaChannel fails with Java 17
- Resolve errors and warnings of maven-javadoc-plugin when using Java 17
- Several errors are logged during start up with Java 17
- Analyse warnings logged by maven-pdf-plugin when using Java 17
- Update Maven plugins to latest versions

- Update Maven dependencies to latest versions

Finally, all identified open tasks were summarized in milestone 6.0 of TEAM Engine.

5.5.3. HomeAssistant-SensorThings

The participants found the OGC SensorThings API Standard usable enough to allow for HomeAssistant-SensorThings to be developed within the time frame of the code sprint, albeit not being feature complete.

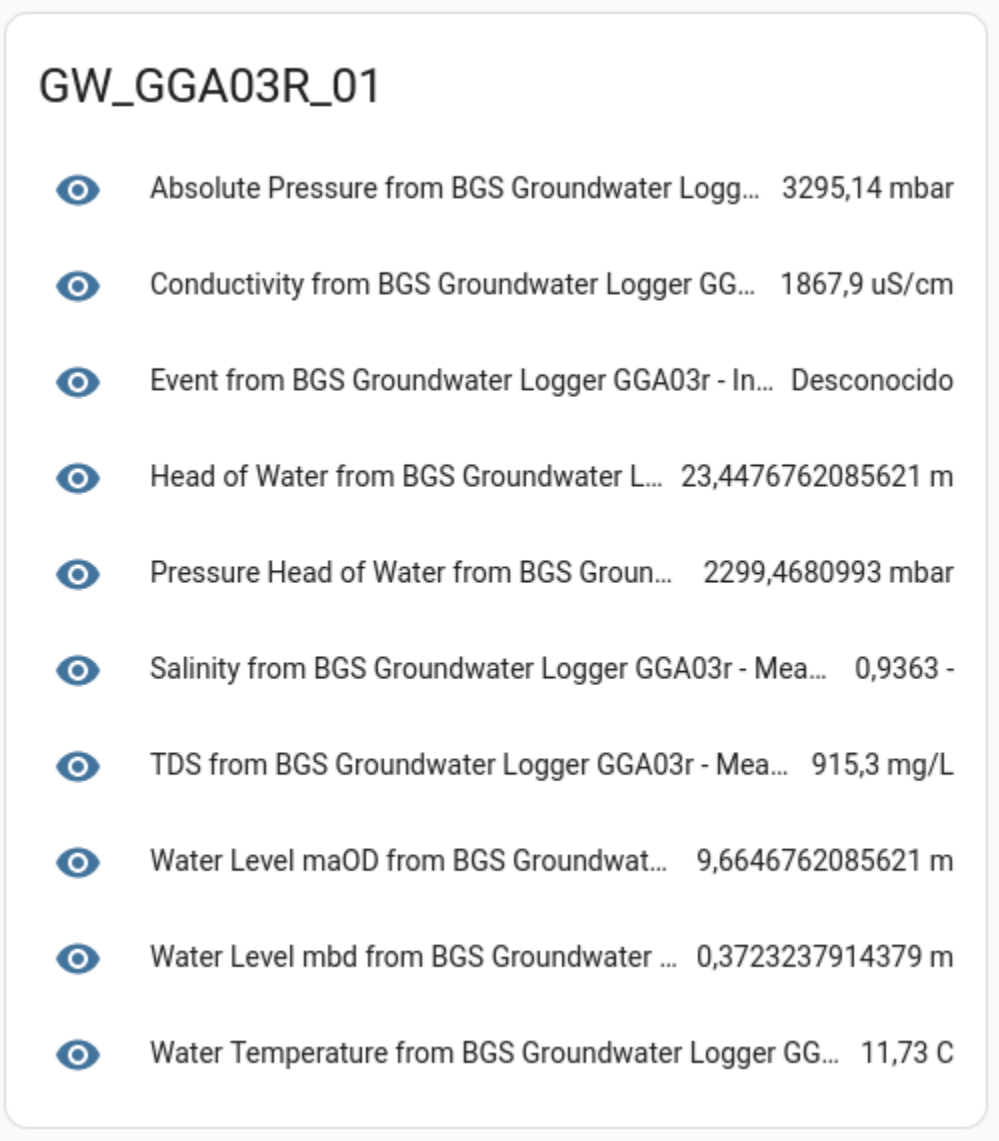

Figure 8 — Partial screenshot of a Home Assistant dashboard showing SensorThings observations from the British Geological Survey, from their FROST server.

The OGC SensorThings API standard has some traits that proved very helpful for the development and debugging of the client: the usage of JSON over HTTP and the availability of full REST-like URIs within the metadata offered by the endpoint.

The nomenclature of concepts posed a minor challenge. An OGC *Datastream* is an HA *sensor*, and a OGC *Thing* is an HA *device* and an OGC API endpoint becomes a *config entry* in HA. The concept of an OGC *Observation* is implicit in the behavior of a HA *sensor*. The OGC concepts of *Sensor*, *Location*, *FeatureOfInterest*, and *ObservedProperty* do not have an equivalent on HA. The HA concepts of *area*, *zone*, *scene*, and others do not have an OGC equivalent.

Participants raised the following concerns during the code sprint.

- Units of measurement are inconsistent across endpoints. A temperature sensor from USGS uses `degC`, one from Fraunhofer uses `°C`, and one from BGS uses `C`. A human will interpret all those as “degrees celsius”, but this mismatch makes automated data comparisons impossible. The standard recommends using UCUM (Unified Codes for Units of Measure) but none of the SensorThings services explored were following this recommendation.
- The values for the description and definition of `ObservedProperty` are unreliable even though they are mandatory fields. All SensorThings services explored were either using empty strings or reusing the name of an `ObservedProperty` as its description and/or definition. No servers produce URIs for the `ObservedProperty` definitions as mandated by the standard.
- Home Assistant focuses on realtime or quasi-realtime data and the only viable strategy for the SensorThings client is to periodically poll for new observations. However, there is no estimation on how often new observations are made, not even a rough one. Some sensors can provide new observations every second, while others might provide observations once every hour or once per day. Polling data each second from a daily sensor is a waste of networking resources whereas polling each day from a secondly sensor introduces an obvious data lag. It’s impossible to choose the right approximate rate of polling. The MQTT extension of the OGC SensorThings API Standard alleviates this issue but complicates the implementation of on some servers and clients.
- The Home Assistant architecture relies on communicating the last known state of a sensor/device/service which is arguably the most common use case for internet-connected sensors. However, SensorThings provides no specific facility for such cases. Instead, the generic way of fetching any and all historical observations must be used, including a query string like `?$top=1&$orderby=phenomenonType desc`. Some of the participants observed that the specification states that «the SensorThings API is designed specifically for the resource-constrained IoT devices and the web developer community» but it can be argued that query languages are ill-suited to be implemented in constrained device.

The OGC SensorThings API Standard is different from other contemporary OGC API Standards (such as Maps, Features, or Coverages) in that it does not implement content negotiation. It would be interesting to consider content negotiation as an extension so the last observation could be fetched in human-readable HTML form or so a series of observations could be downloaded as a machine-readable CSV.

Finally, it would be possible to implement a SensorThings *server* on top of a Home Assistant installation. The scope of such development falls out of a 3-day code sprint, but could be achieved in a time frame of several weeks. This would potentially provide a cost-effective way to provide SensorThings endpoints for domestic-grade sensor platforms such as weather stations or soil moisture sensors.

5.5.4. Maplibre

The rendering pipleline is generally not exposed to the user so the participants working on Maplibre experimented with accessing the rendering pipeline. They implemented functionality to export the depth buffer (shown in the top left of Figure 9). The approach offers the ability to create light effects and special environmental effects such as an atmospheric haze (shown in the bottom right of Figure 9). The participants configured their prototype so that when the viewing point is far away the effect is disabled. The participants also examined other parts of the pipeline so that any object that is rendered is based on the modified depth buffer. This included, for example, modification of the coordinate buffer (shown in the bottom left of Figure 9) so that tiles at the top of the screen represent a large space. The third aspect that can be modified using this approach is the color of the image, for example to enable a user to adjust the brightness or to create aesthetic visuals (shown in the top right of Figure 9).

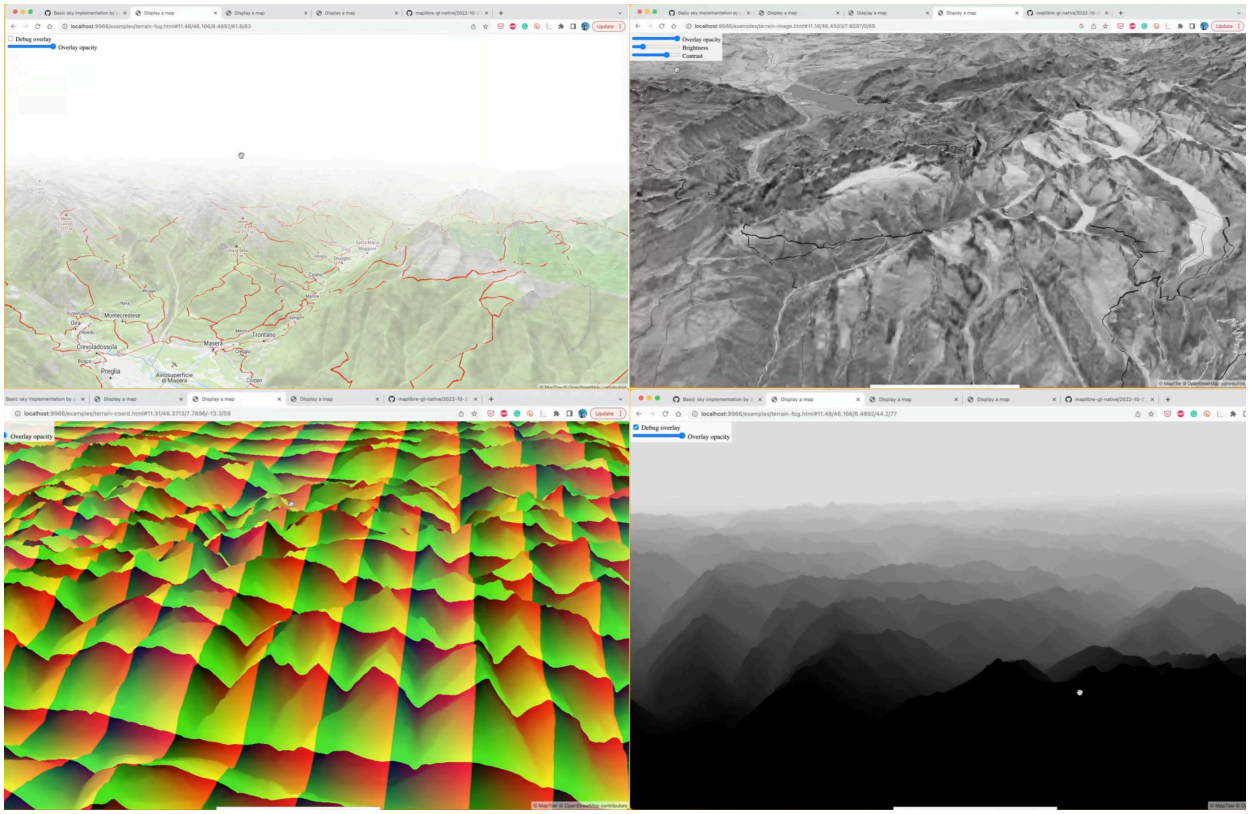

Figure 9 — Screenshot of Maplibre demonstration

5.5.5. MDME

The MDME project uses vjsf internally to render a web based metadata edit form derived from a json schema. A visual edit form for MCF has proven to be useful in yaml oriented metadata workflows for some user groups. Some json schema concepts such as `any-of` are not supported by vjsf. During the code sprint MDME was updated to prevent the use of `any-of` and selector patterns having '*'. Also during the code sprint, import of metadata from ISO 19139 records (as discussed in this GitHub Issue) and export to ISO 19139 records was introduced, as well as export to STAC and OGC API — Records collections (based on a transformation service provided by pygeoapi/pygeometa).

5.5.6. OL-Cesium

During the code sprint, the participants fixed support for the latest Cesium version 1.104. Several of the examples were fixed. Another fix updated the code base so that the Parcel bundler could support better reading of the code base. Support for ECMAScript 6 (ES6) exports was also fixed. The participants also worked on the destruction method to improve memory handling. The participants also worked on the migration to Typescript. Another achievement was the work done on adding polygon extrusion support. Another feature that was implemented during the code sprint was OL-Cesium support in Geomapfish.

5.5.7. OSGeo pygeometa

The pygeometa project added support for detailed schema capability reporting so that a client can be aware of which schemas are supported, and whether those schemas have read and/or write support. This was also implemented as a pygeoapi plugin and made available as part of the pygeoapi public demo. "pygeometa as a service" is used by the live demo of the MDME project. The schema capability integration was added by MDME as part of the code sprint.

5.5.8. Smapshot

A screenshot of the application, shown during the code sprint, is presented in Figure 10. As illustrated, Smapshot enables users to geolocalise historical images in 3D.

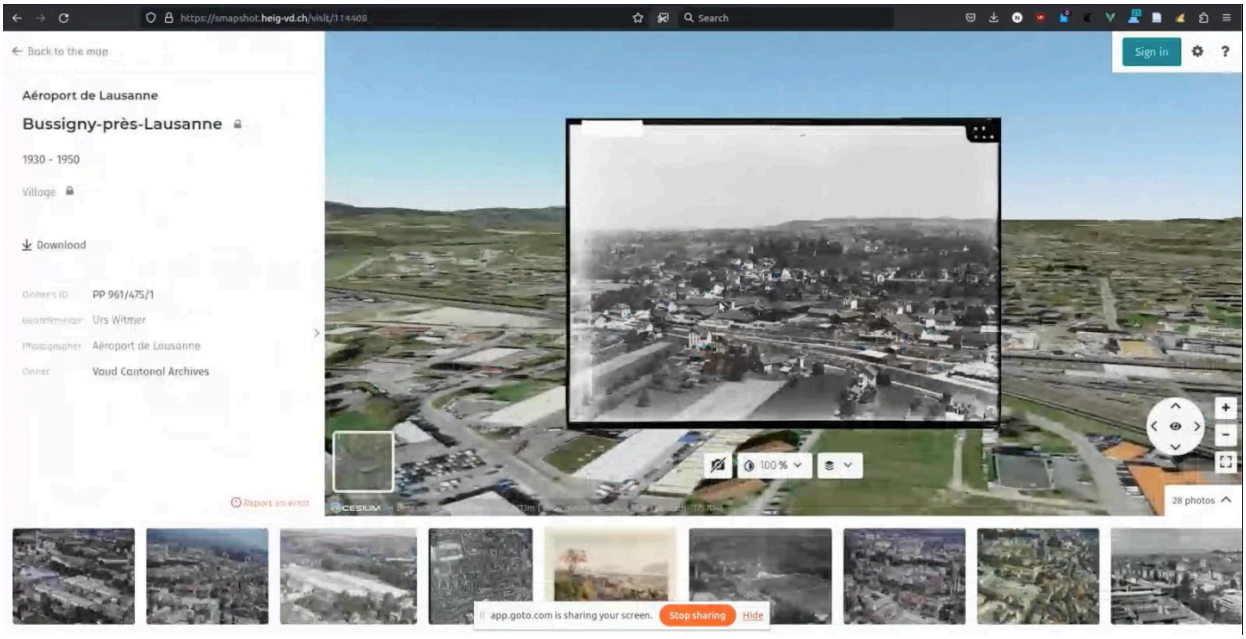

Figure 10 — A screenshot of Smapshot

During the code sprint the OGC GeoPose Draft Standard was tested as a way to return position and orientation data of the 3D geolocalized images from smapshot from an OGC API — Features implementation. Support for GeoPose-encoded data has been added to the Open Source NodeJs server of OGC API features: CodeSprint PR. A quick data visualization has been implemented in Cesium to display the geopose of the camera retrieved from the API with smapshot data and the geopose of the photograph. A screenshot of the application is shown in Figure 11.

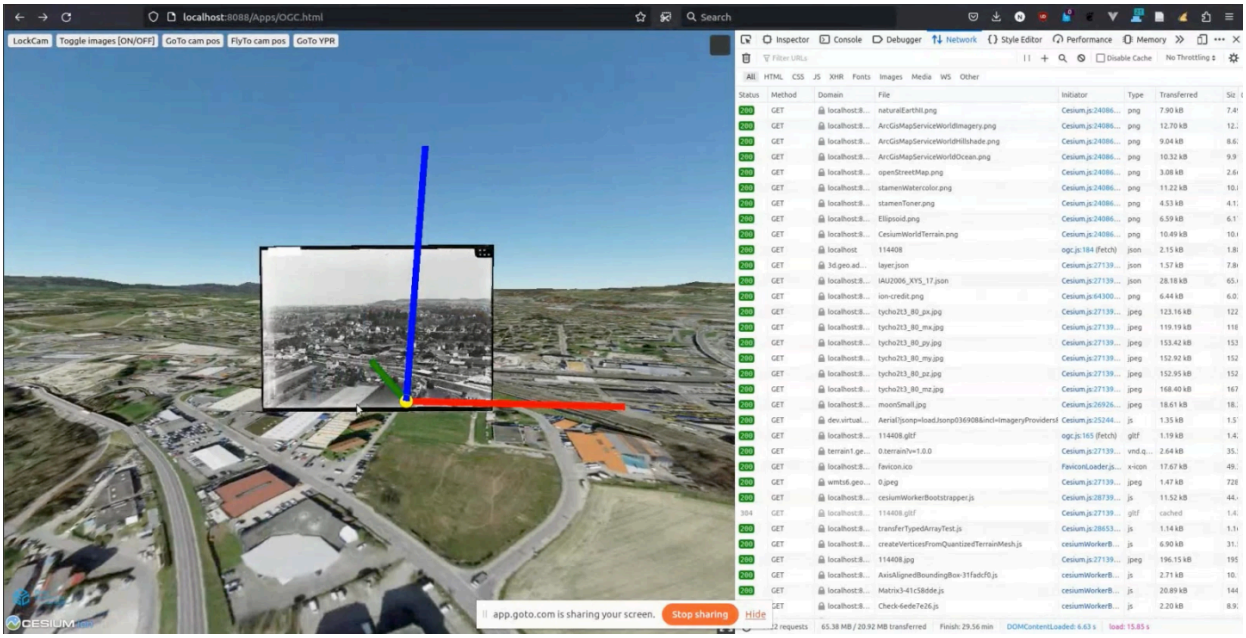

Figure 11 — Position and orientation from a geopose represented in Smapshot

5.5.9. go-ogc

Work was started to add OGC API — Records support to the library. This involved adding an extension mechanism to the existing OGC API — Features structs, in the same way that the OGC API — Records candidate Standard extends the OGC API — Features Standard. The library is being used at Planet Labs where the OGC API — Features Standard is being used to provide metadata about Planet's imagery catalog. The records branch of the go-ogc library is being used to test an update to the internal service that adds support for OGC API — Records.

5.5.9.1. Suggestions

5.5.9.1.1. Add `conformsTo` property to OGC API — Features

The OGC API — Records specification extends OGC API — Features by adding a `conformsTo` property to GeoJSON records (or features) which would be a useful extension point for other specifications in addition to OGC API — Records (e.g. STAC API or other standards that might extend Features). Instead of being defined in the OGC API — Records candidate Standard, it would be useful if the OGC API — Features Standard defined the `conformsTo` property as a common way to advertise conformance with other specifications that might extend Features.

5.5.9.1.2. Weigh the pros/cons of adding top-level GeoJSON Feature members

The OGC API — Records specification adds top-level members to GeoJSON Features. For example, the `time` member is added at the top level of the Feature object. By contrast, members like `created` and `updated` are added to the `properties` object. The location of these members will determine how they can be accessed by existing software that works with GeoJSON. A suggestion would be to evaluate the pros and cons of adding top-level members to GeoJSON Features and to consider whether it would be better to add these members to the `properties` object instead.

For example, using Fiona (which delegates to OGR/GDAL for feature access), the `time` member cannot be accessed when reading a feature that looks like this:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [0, 0]
  },
  "properties": {
    "name": "Null Island"
  },
  "time": "2023-05-10T13:00:00Z"
}
```

GeoJSON Feature with top-level time member

By contrast, the `time` member can be accessed when reading a feature that looks like this:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [0, 0]
  },
  "properties": {
    "name": "Null Island",
    "time": "2023-05-10T13:00:00Z"
  }
}
```

GeoJSON Feature with time in the properties object

The same limitation applies to mapping clients that might allow filtering or styling features based on feature properties. In these cases the top-level members are not useful. While updates to these existing libraries can be made to handle this new extended type of GeoJSON, it is likely to slow adoption, and the spec may find more interoperability if it were designed to work with existing libraries.

5.5.10. GPQ

The GQP source was recently open sourced and is now available under the Planet Labs GitHub organization: https://github.com/planetlabs/gpq. A CI workflow was added to publish a WASM binary of the utility to GitHub Pages, allowing conversion of GeoJSON to GeoParquet and vice versa in the browser. In addition, work was done to improve how the Parquet schema is derived from GeoJSON features. Previously, the schema would be derived from the first feature in a collection with all non-null property values. Now, the schema is incrementally built over a range of features, allowing for a more complete schema to be derived. This is especially useful for large collections with sparse properties.

6

DISCUSSION

6 DISCUSSION

6.1. IOGP GIGS tests and the OGC GeoAPI Standard

The Geospatial Integrity of Geoscience Software (GIGS) tests are an open-source digital testing framework for evaluating the capability of software in establishing and maintaining the integrity of geospatial data. The tests are developed and maintained by the International Association of Oil & Gas Producers (IOGP). The GIGS tests can be found on the gigs.iogp.org website.

The participants from Geomatys worked on PROJ-JNI to enable PROJ-JNI to execute additional GIGS tests. PROJ is a coordinate transformation software package, built using C/C++ library, that transforms geospatial coordinates from one CRS to another. PROJ-JNI is a Java Native Interface for PROJ.

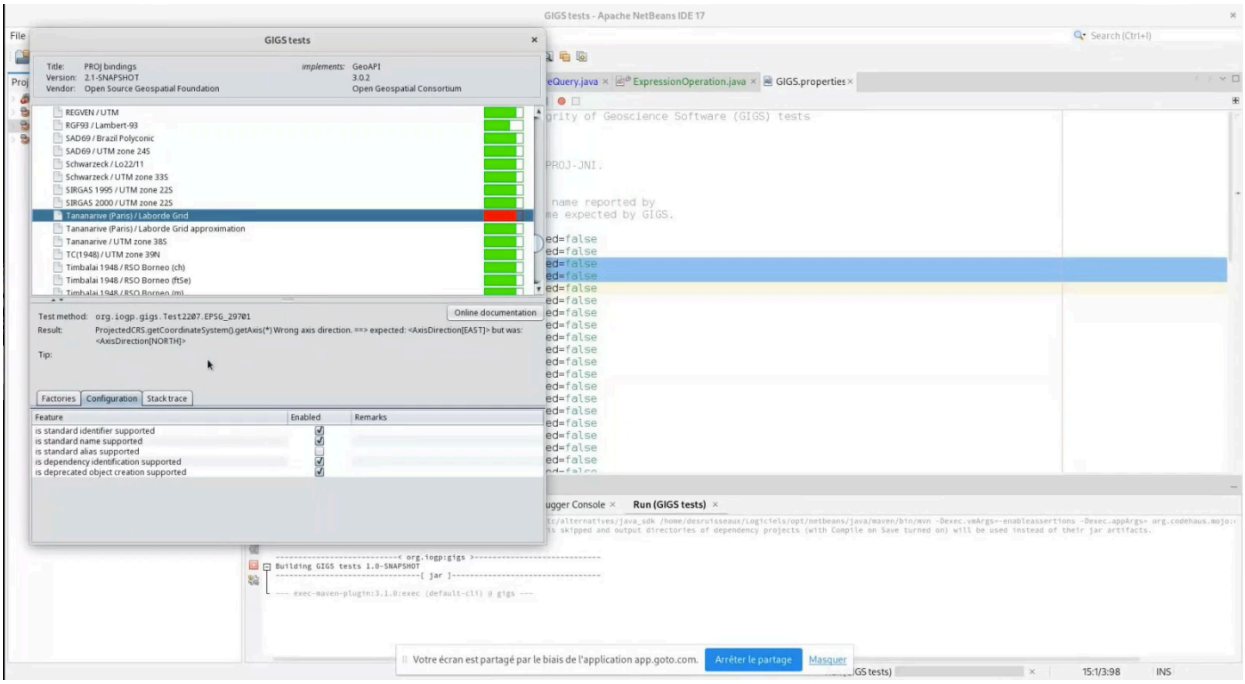

Figure 12 — Example execution of the GIGS test sofware

Not every software product will support every feature that the GIGS tests specify, so a properties file is provided to enable the user to enable or disable specific aspects of the tests. PROJ-JNI is an implementation of the OGC GeoAPI Standard.

6.2. Home Assistant and OGC SensorThings API

HomeAssistant-SensorThings was introduced during the code sprint. The sprint participants implemented an ability for HomeAssistant-SensorThings to automatically and periodically fetch observations from an instance of the OGC SensorThings API. The software was also enabled to offer automatic discovery of 'Things' as HomeAssistant devices and 'Datastreams' as HomeAssistant sensors.

One of the challenges encountered was that there is some inconsistency in units of measurement between specific implementations of the OGC SensorThings API. This issue is explained further by Figure 13 which shows in Panel 1 an implementation that uses `degC`, and in Panels 2 an implementation that uses `°C` and in Panel 3 an implementation that uses `C` to represent temperature in degree Celsius. It is also worth noting the difference in the values of the definition properties in the implementations shown in Panels 2 and 3. This difference could be mistakenly interpreted to mean that the observed properties are semantically different. This issue could potentially be addressed by providing examples of the use of OGC Rainbow (formerly known as the OGC Definitions Server) to support the representation of units of measure in sensor-related standards.

Figure 13 — Differences in Units of measure representation in a sample of SensorThings API instances

Another observation was although the data type of the definition property is supposed to be a URI, a number of the implementations appeared not to specify URIs in this property. The sprint participants therefore recommended the creation and maintenance of a taxonomy of well-known observable properties, including their JSON representations containing URIs. Such a capability could, for example, be implemented as an extension of OGC Rainbow.

Another observation was that there was no indication of the rate of observation. This means it becomes unclear for some client applications how long they should wait before retrieving the next observation. The absence of an indicator of the rate of observation is partly because

SensorThings API uses a publish/subscribe mechanism through MQTT and thus does not rely on polling. However, not all client applications support MQTT. The participants noted that OGC API — Records has a resolution property that helps client applications determine how often to request observations.

The sprint participants also noted that there is no shorthand for fetching the last observation. The participants noted that although such a query could be constructed using ODATA query options like `?$top=1&$orderby=phenomenonTime desc`, such a capability would have to be implemented with caution because some constrained devices might not be able to support querying of very large data collections.

A content-type negotiation mechanism that enables a client application to retrieve content in HTML, JSON, or another language was also noted as a potential enhancement for the SensorThings API standard.

6.3. IndoorGML

Participants from IIT Bombay reviewed the IndoorGML standard as preparation for future contribution towards development of e-learning materials. This OGC® IndoorGML Standard specifies an open data model and XML schema for indoor spatial information. IndoorGML is an application schema of the OGC Geography Markup Language (GML) Standard.

The participants identified the following potential issue. When there are different sizes of walls in a room, how are cells assigned? For example, considering an office area where a hall is divided into different cubicles. The cubicles are created with the help of temporary partitions which are not of the height of the wall. Each cubicle is allocated for a designated task and needs a unique location point. Since the cubicle is not divided by walls there is no unique distinction for that particular cubicle. In this case, how can the temporary walls creating the cubicles be marked and a specific location be created for the cubicle so that navigation to that cubicle is feasible? This issue requires clarification in future versions of IndoorGML.

6.4. GeoXAML 3.0

This code sprint activity demonstrated the basics for enabling powerful Attribute Based Access Control including geospatial conditions. Regarding a production solution, the request rewriting to inject (not only) geospatial conditions via the OGC Filter parameter can be considered high-performance: little information is required to create a request in the PEP and send it to the GeoPDP. Likewise, the response from the GeoPDP to the PEP includes a decision and zero to multiple obligations that control the construction of the Filter. The deployment of the PDP is mission critical, but because this backend service is stateless, scaling is “straight forward” in cloud infrastructures.

A successful demonstration was given during the last day brief back. The QGIS application was used to connect to the GeoServer WFS using the regular OGC WFS 2.0 service. Invisible to the

client, the PEP modified the WFS request sent by QGIS based on the access rights expressed in the GeoXACML policy.

6.5. pygeoapi

The pygeoapi work involved integration with a new data provider to expose scientific and ocean data from a specific API. Implementation of the new data provider had been done prior to the code sprint and some final integration work had been completed during the code sprint. The data provider was enabled to access ERDDAP, a server technology by the National Oceanic and Atmospheric Administration(NOAA). ERDDAP is used by several of NOAA's partners to disseminate real time scientific data. Servers supporting ERDDAP can be accessed through an interface that supports JSON messages, as well as other approaches. The sprint participants working on pygeoapi wanted to expose an ERDDAP instance through an interface that implements the OGC API — Features Standard.

An implementation of the prototype which has been built using pygeoapi was demonstrated during the code sprint. The demo assumes that there is a data provider upstream that is producing observations from which the alerts are derived. From there the alerts are created and pushed into an implementation of OGC API — Features — Part 4: Create, Replace, Update and Delete. The OGC API — Features implementation sends a message to a broker. In addition to the features being inserted into the collection, a notification would then be sent out to subscribers. The implementation uses a broker called Mosquito, which supports MQTT. The client application was implemented using OWSLib.

Accessing the HTML-encoded AsyncAPI definition document, leads to a view such as the one shown in Figure 14. Note that the document advertizes an MQTT server. In addition to MQTT, AsyncAPI also supports other protocols such as Kafka and AMQP.

Figure 14 — Screenshot of a rendered HTML-encoded pygeoapi asyncapi document

Subscription information can be obtained from the description of a feature collection that supports the event-driven mechanism. An example link for subscribing to notifications through MQTT is shown in the following listing:

```
{
    "id": "obs",
    "title": "Observations",
    "description": "Observations",
    "keywords": [
        "observations",
        "monitoring"
    ],
    "links": [
        {
            "type": "application/json",
            "rel": "root",
            "title": "The landing page of this server as JSON",
            "href": "https://demo.pygeoapi.io/master?f=json"
        },
        {
            "type": "application/json",
            "rel": "self",
            "title": "This document as JSON",
            "href": "https://demo.pygeoapi.io/master/collections/obs?f=json"
        },
        {
            "type": "application/geo+json",
            "rel": "items",
            "title": "items as GeoJSON",
            "href": "https://demo.pygeoapi.io/master/collections/obs/items?f=
json"
        },
        {
            "type": "application/json",
```

```
            "rel": "items",
            "title": "Subscription information (Pub/Sub)",
            "href": "mqtt://localhost:1883",
            "channel": "collections/canada-surface-weather-obs"
        }
    ],
    "extent": {
        "spatial": {
            "bbox": [
                [
                    -180,
                    -90,
                    180,
                    90
                ]
            ],
            "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
        },
        "temporal": {
            "interval": [
                [
                    "2000-10-30T18:24:39+00:00",
                    "2007-10-30T08:57:29+00:00"
                ]
            ]
        }
    },
    "itemType": "feature",
    "crs": [
        "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
    ],
    "storageCRS": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
}
```


The listing above highlights a need to be able to include additional parameters in links. Since RFC 8288 identifies a specific set of link-params that are allowed in links, this raises the question of whether inclusion of the channel parameter would be allowable per the RFC. If the RFC does not allow additional parameters, then how might the need to advertize channel endpoints be addressed in a future Pub/Sub conformance class of OGC API Standards? These questions were posted to the OGC API — Common repository for discussion by the SWG. A screenshot of MQTT Explorer receiving notifications from an OGC API — Features implementation is presented in Figure 15. MQTT Explorer is an MQTT client that displays an overview of MQTT topics and supports other tasks involving MQTT.

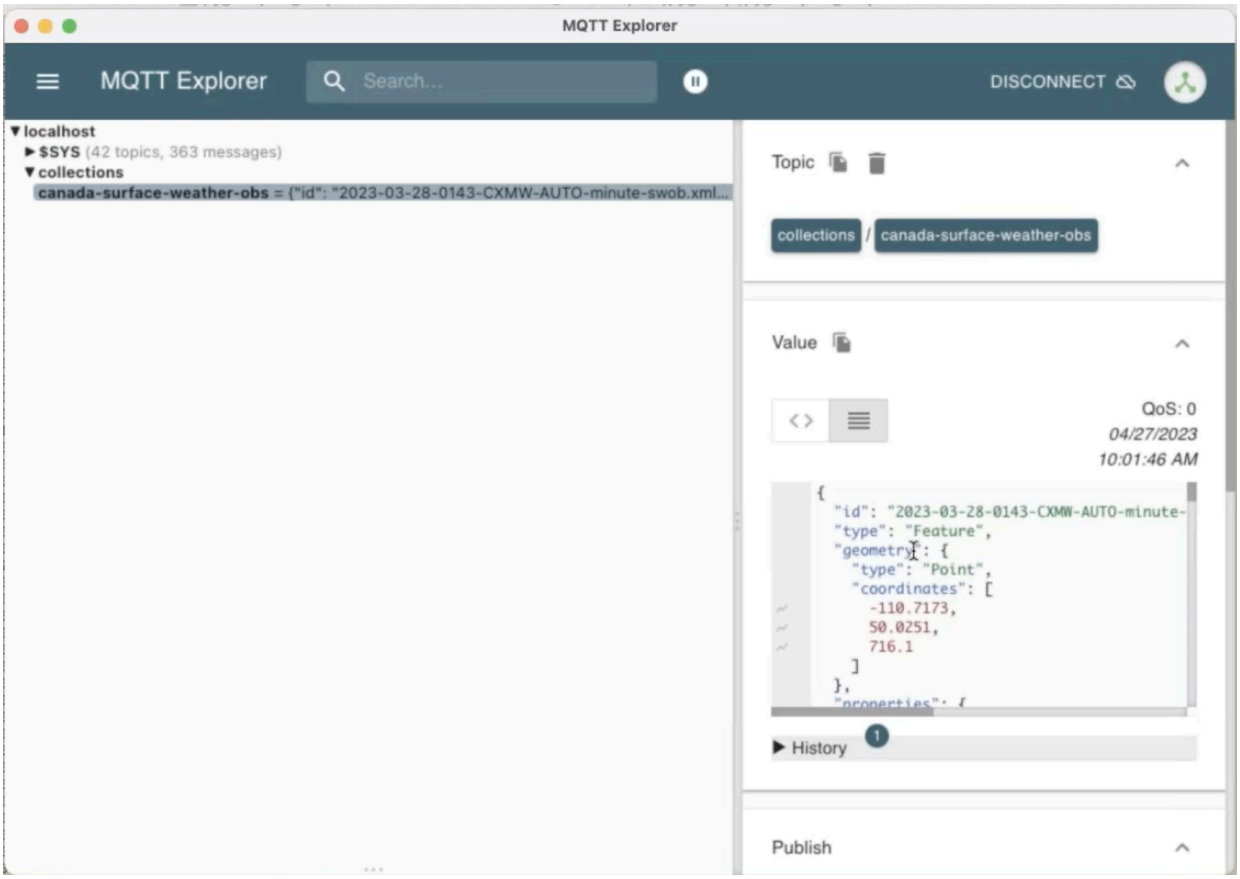

Figure 15 — Screenshot of MQTT Explorer receiving notifications from an OGC API - Features implementation

The experimentation is envisaged to contribute to development of the WIS2 framework by the World Meteorological Organization (WMO). A screenshot of a WIS 2.0 prototype receiving notifications through the Pub/Sub mechanism is shown in Figure 16.

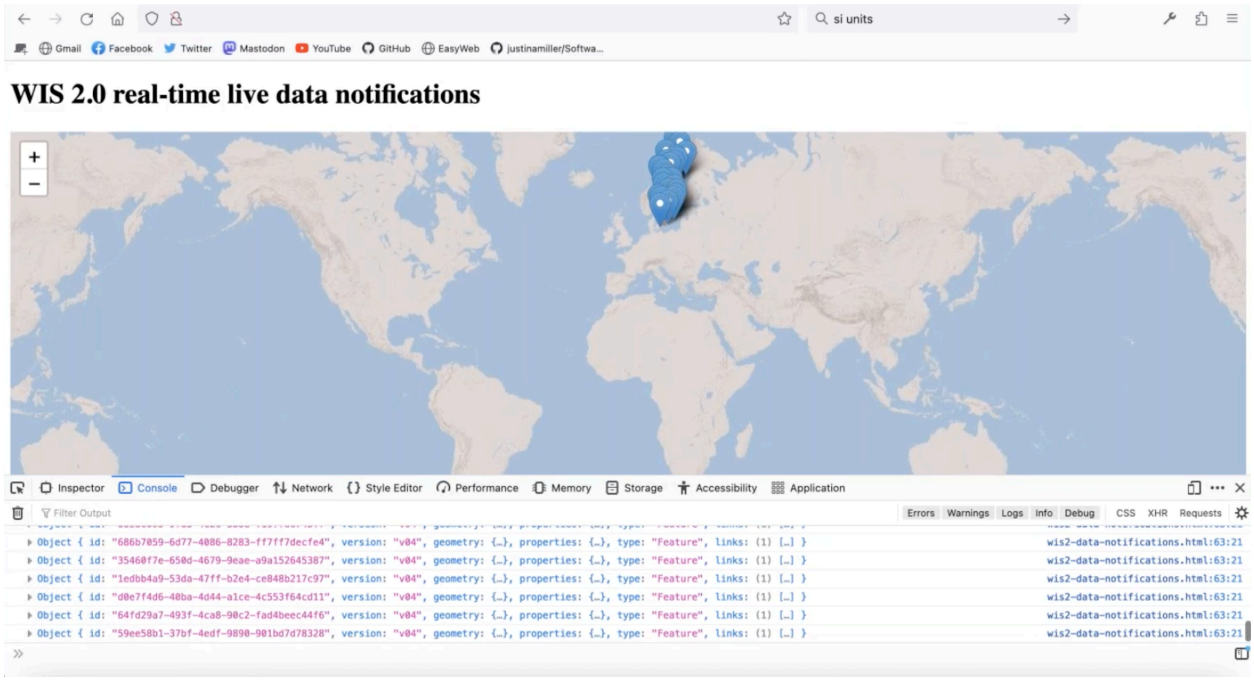

Figure 16 — Screenshot of a WIS 2.0 prototype receiving notifications through a Pub/Sub mechanism

6.6. Styling in OpenLayers

OpenLayers has long had support for letting users create a style. Users can change the stroke or color of a symbol. The benefit is that it is flexible. The downside is that it needs to run on the CPU in the context of the user's client-side environment. The solution is styles that can be passed to a different context, for example to a worker. An example of the previous approach is shown below.

```
function style(feature) {
    if (feature.get('type')) ≡ 'highway') {
        return new Style({
            stroke: new Stroke({
                color: userColor,
                width: 10,
            }),
        });
    }

    return new Style({
        stroke: new Stroke({
            color: userColor,
            width: 2,
        }),
    });

}
```

An example of a new approach that was prototyped during the code sprint is shown below. The style is represented as a Flat style with expressions that can be supported by a WebGL canvas. One limitation is that while it is possible to create complex styles with this syntax, it can result in a lot of redundancy as multiple symbolizer properties are applied to a single “class” of features.

```
{
    'fill-color': 'orange',
    'stroke-width': ['match',['get','type'],'highway', 10, 2],
    'stroke-color': ['var','userColor']
}
```

This new approach is already supported by OpenLayers for raster rendering, as demonstrated by Figure 17 which shows a screenshot of an OpenLayers application that has modified the display of one of the bands of a raster image on the client-side.

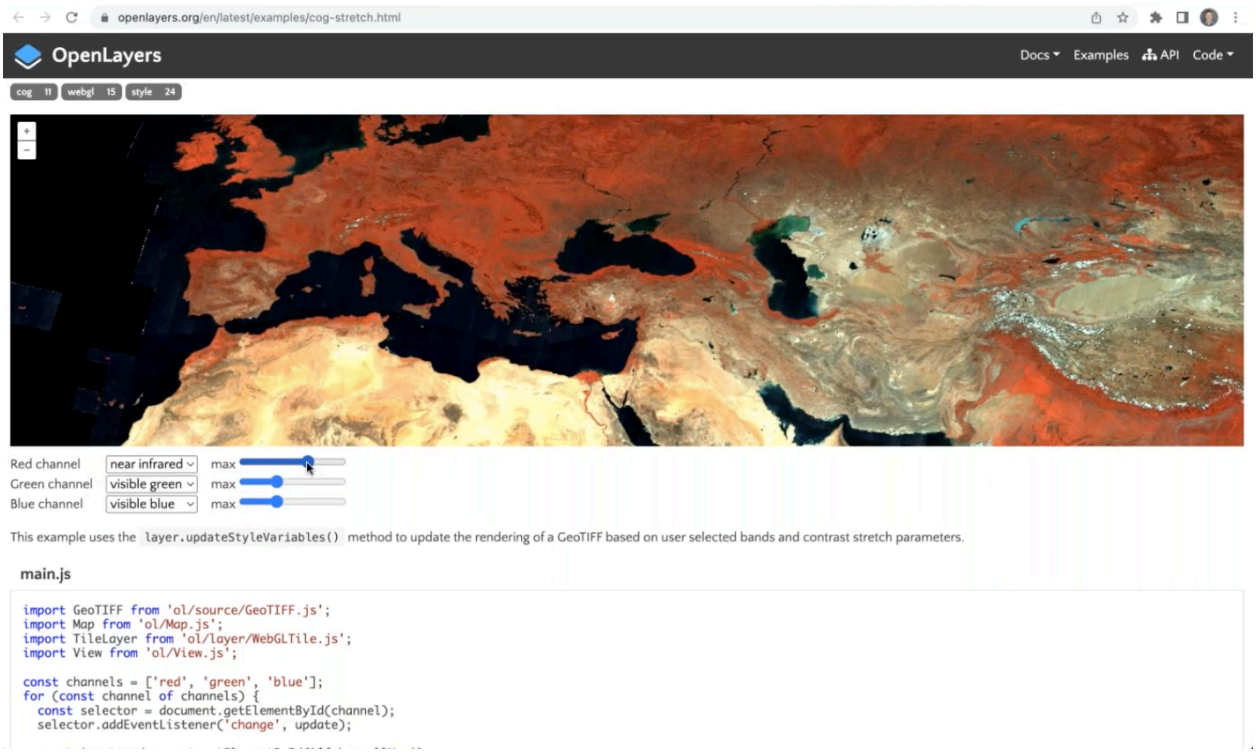

Figure 17 — A screenshot of an OpenLayers application that supports client-side modification of styles

Currently OpenLayers does not support expressions in the use of this new approach for representing styles. During the code sprint OpenLayers contributors discussed support for expressions. An example of how the approach could be applied to vector styling is shown below.

```
[
 {
    filter: ['=', ['get', 'type'],'highway'],
    style: {
        'stroke-width': 10,
        'stroke-color': 'orange',
    },
 },
 {
    filter: ['=', ['get', 'type'], 'dirt road'],
    style: {
```

```
        'stroke-width': 2,
        'stroke-color': 'brown',
    },
 },
]
```

6.7. go-ogc

Participants from Planet Labs worked on an implementation of go-ogc that would support OGC API — Records in the future. Currently the software supports STAC as illustrated by Figure 18. Ongoing work within OGC to align OGC API — Records has therefore created an opportunity for STAC implementations to also support OGC API — Records which is made possible, partly because both STAC and OGC API — Records extend OGC API — Features.

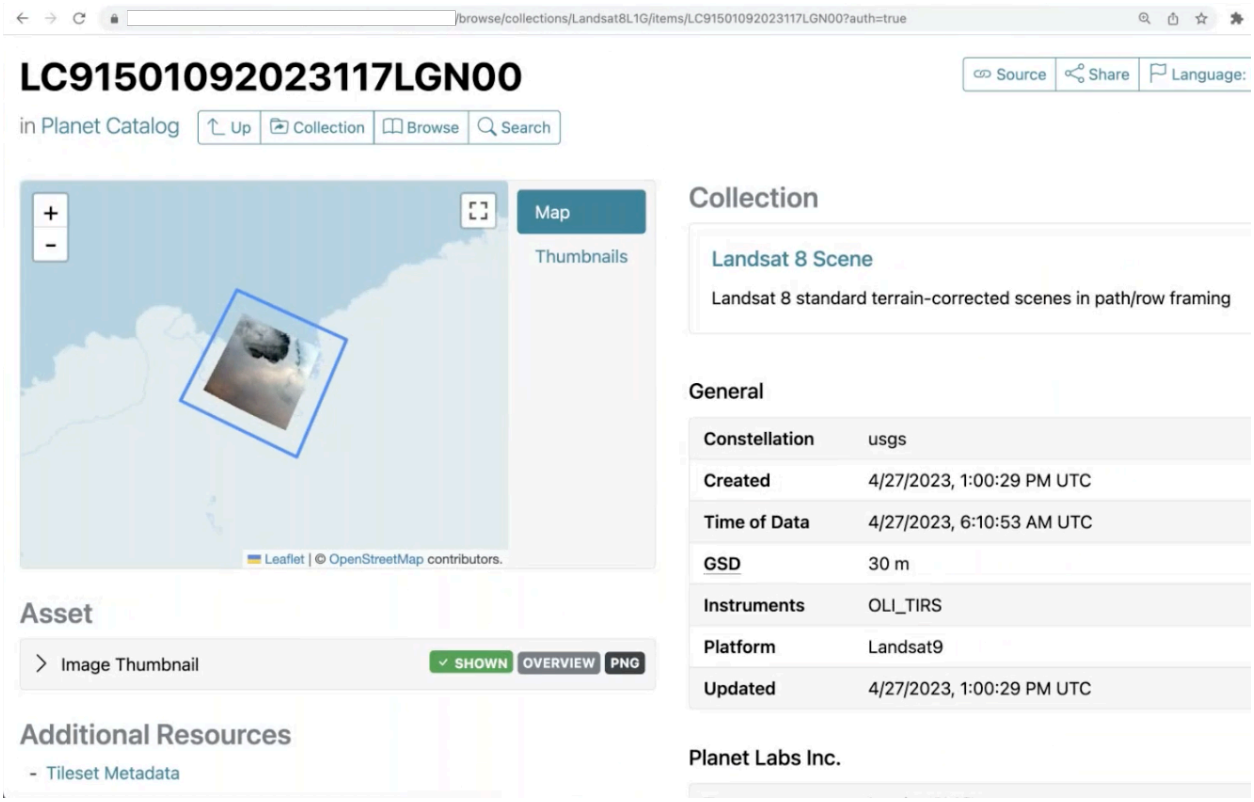

Figure 18 — Screenshot of the Planet Catalog interface

Implementation of support for OGC API — Records in go-ogc highlighted the potential for supporting other JSON-based formats specified by OGC Standards. For example, GeoPose could be used to represent the position and orientation of cameras used by Earth Observation platforms. Another format that could be supported, specifically for representing imagery metadata, is GMLJP2.

7

CONCLUSIONS

7 CONCLUSIONS

The code sprint provided an environment for the development and testing of prototype implementations of open standards, including implementations of draft and approved OGC Standards. Furthermore, the code sprint enabled the participating developers to provide feedback to the editors of OGC Standards. In addition, the collaborative environment provided by the code sprint enabled developers to fix open issues in products, develop new features, improve documentation, improve interoperability with other libraries/products, and develop prototype implementations of OGC standards. This engineering report therefore concludes that the code sprint met all of its objectives and achieved its goal of accelerating the support of open geospatial standards within the developer community. The following subsection presents recommendations for future work.

7.1. Future Work

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

- experimentation and prototyping of implementations of OGC Styles and Symbology;
- provide examples of the use of OGC Rainbow (formerly known as the OGC Definitions Server) to support the representation of units of measure, in sensor-related standards;
- experimentation and prototyping of implementations of OGC GeoXACML;
- advance the development of an Executable Test Suite for OGC API — Tiles;
- advance the development of an Executable Test Suite for OGC API — Records; and
- advance the development of an Executable Test Suite for OGC GeoXACML.

A

ANNEX A (INFORMATIVE) REVISION HISTORY

A ANNEX A (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2023-03-11	0.1	G. Hobona	all	initial version
2023-04-29	0.2	G. Hobona	all	Updated with feedback from participants
2023-06-29	0.3	G. Hobona	all	Editorial changes

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] OGC: OGC 21-008: Joint OGC OSGeo ASF Code Sprint 2021 Summary Engineering Report, 2021

[2] Aleksandar Balaban: OGC 21-019, OGC Testbed-17: Attracting Developers: Lowering the entry barrier for implementing OGC Web APIs. Open Geospatial Consortium (2022). https://docs.ogc.org/per/21-019.html